Software Engineering Institute

Software Assurance Curriculum Project Volume III: Master of Software Assurance Course Syllabi

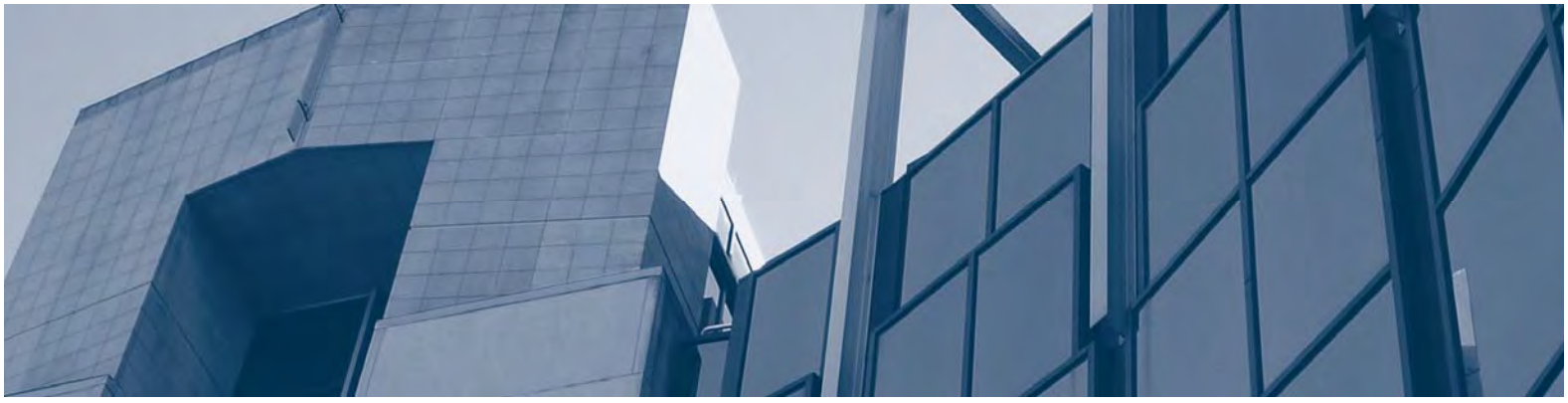
Nancy R. Mead, Software Engineering Institute
Julia H. Allen, Software Engineering Institute
Mark Ardis, Stevens Institute of Technology
Thomas B. Hilburn, Embry-Riddle Aeronautical University
Andrew J. Kornecki, Embry-Riddle Aeronautical University
Richard C. Linger, Oak Ridge National Laboratory

March 2011; Revised July 2011

TECHNICAL REPORT
CMU/SEI-2011-TR-013
ESC-TR-2011-013

CERT® Program

http://www.sei.cmu.edu
http://www.cert.org



Carnegie Mellon

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about SEI publications, please visit the library on the SEI website (www.sei.cmu.edu/library).

Table of Contents

List of Tables

Acknowledgments

The authors thank the following individuals for their contributions to this report. We greatly appreciate their insights and efforts.

- Our sponsor Joe Jarzombek, U.S. Department of Homeland Security (DHS) National Cyber Security Division (NCSD), had the insight to recognize the need for such a curriculum and support its development.
- These individuals provided careful reviews and helpful feedback:
  - Christopher Alberts, Carnegie Mellon University, Software Engineering Institute, CERT Program
  - Alan Hevner, Citigroup/Hidden River Chair of Distributed Technology Information Systems Decision Sciences Department, University of South Florida
  - Lori Kaufman, Carnegie Mellon University, Software Engineering Institute, CERT Program
  - Lisa Young, Carnegie Mellon University, Software Engineering Institute, CERT Program
  - Nick Brixius, Professor of Software Engineering, Embry-Riddle Aeronautical University
  - Remzi Seker, Associate Professor and Chair of Computer Science, University of Arkansas, Little Rock
  - Bob Ellison, Carnegie Mellon University, Software Engineering Institute, CERT Program

We would also like to acknowledge the contributions of our editors, Pennie Walters and Melanie Thompson, as well as those who commented during the public review process.

How to Use This Document

The syllabi in this document were created to support the development of a set of courses to be used in a master of software assurance curriculum, as described in Appendix F of *Software Assurance Curriculum Project Volume I: Master of Software Assurance Reference Curriculum* [Mead 2010], henceforth referred to as *Volume I*. An alternate approach is to integrate selected course content into an existing master of software engineering program. Course designers using these syllabi should become familiar with the contents of *Volume 1* and rely on it as a primary resource for software assurance course design.

Each syllabus contains the following components: a catalog description, the course prerequisites and corequisites, expected student outcomes, a list of topics, a set of primary and secondary sources, descriptions of assignments and in-class activities, and a suggested schedule. The following sections explain some of these components.

Prerequisites and Corequisites

The designated prerequisites and corequisites are based on the assumption that all the courses are offered as part of a master of software assurance program that follows the guidance in *Volume I* [Mead 2010]. These requisites are designed to ensure that the expected student outcomes can be achieved. If all the courses are not offered or they are offered in a sequence that is different from that described in the prerequisites/corequisites, course designers will have to either use other courses in their curriculum that have similar content or designate alternative prerequisites/corequisites to ensure adequate knowledge and experience. If appropriate courses are not available, the expected student outcomes will likely need to be modified.

Expected Student Outcomes

The course outcomes are described in Appendix F of *Volume I*. Along with the list of topics, the outcomes are derived from the MSwA2010 Body of Knowledge (BoK), which is contained in *Volume I* and provided in Appendix A of this document for easy reference. The curriculum outcomes listed in Chapter 4 of *Volume I* were the primary influence on the organization and content of the MSwA2010 BoK, so, collectively, the course syllabi student outcomes represent the MSWA2010 outcomes. For ease of use, Appendix B provides a table that indicates which knowledge areas of the MSwA BoK are covered by which courses in this syllabi. Therefore, course designers modifying the syllabi must ensure that overall curriculum outcomes are not adversely affected and that, when necessary, appropriate alternative outcomes are developed.

Primary and Secondary Sources

The primary sources are recommended both as the main sources that course designers would use when developing course materials and as student reading material. In many cases, other sources can provide similar material, but it is recommended that course designers examine the primary sources as candidates.

Secondary sources are listed to provide additional background and resources that course designers might wish to use but that are not considered as comprehensive or as broadly applicable as the

primary sources. The secondary sources include journal papers, web sources, and standards. Such secondary sources might be used as reading assignments on specific topics or as support for student project work. Appropriate secondary sources will change over time as software technologies and educational needs continue to evolve.

Assignments and In-Class Activities

Assignments and in-class activities include regular reading assignments and individual and group exercises. Most courses include a student team project that represents a major course learning activity. These assignments, activities, and projects are all elaborated in the suggested schedule. This is certainly an area where course designers may want to introduce alternatives based on instructor experience and interest, special domains emphasized in a particular program, and available technology and tools.

Suggested Schedule

The suggested schedule outlines a week-by-week schedule of topics and activities that portray one approach for achieving course objectives. All syllabi assume a semester-long, 14-week schedule with one session per week. Each session typically requires three hours of student effort but could be divided into multiple shorter sessions, depending on the program teaching plan.

Course designers may want to consider an alternate schedule of weekly activities—a different ordering of topics and alternate reading assignments, exercises, and projects. It is recommended that any redesign should use the expected student outcomes as a guide and checklist.

Abstract

Modern society depends on software systems of ever-increasing scope and complexity in virtually every sphere of human activity including business, finance, energy, transportation, education, communication, government, and defense. Because the consequences of failure can be severe, dependable functionality and security are essential. As a result, software assurance is emerging as an important discipline for the development, acquisition, and operation of software systems and services that provide requisite levels of dependability and security.

This report, the third volume in the Software Assurance Curriculum Project sponsored by the U.S. Department of Homeland Security, provides sample syllabi for the nine core courses in the Master of Software Assurance Reference Curriculum. That curriculum, detailed in Volume I, Master of Software Assurance Reference Curriculum (CMU/SEI-2010-TR-005), presents a body of knowledge from which to create a Master of Software Assurance degree program, as both a stand-alone offering and as a track within existing software engineering and computer science master's degree programs. Volume II, Undergraduate Course Outlines (CMU/SEI-2010-TR-019), presents seven course outlines that could be used in an undergraduate curriculum specialization for software assurance.

This volume is part of our transition plan for assisting educators who wish to implement a Master of Software Assurance degree program, specialization, or certificate program. In addition to application in a standard university program, these syllabi may also be useful for educators developing courses for industry practitioners.

1 Introduction

This report provides sample syllabi for the nine core courses in the Master of Software Assurance Reference Curriculum [Mead 2010]. We recommend that readers familiarize themselves with the reference curriculum prior to using this document, in order to get a fuller understanding of the context. The syllabi are written in a standard way to include a catalog description, the course prerequisites and corequisites, expected student outcomes, a list of topics, a set of primary and secondary sources, descriptions of assignments and in-class activities, and a suggested schedule. Some of the material is repeated in each syllabus so that each one is self-contained. As noted in the reference curriculum report, each course is intended to be a one-semester course. Although we have suggested prerequisites and corequisites, we recognize that different universities may handle prerequisites in various ways and may use prior course work, work experience, or standardized tests to satisfy prerequisites.

Universities that intend to implement such a curriculum might consider adding one to two courses at a time to work their way up to the full degree program, specialization, or certificate program. When starting out, it would be best to offer a course that does not have a prerequisite within the program, such as the System Operational Assurance or Assured Software Development 1 course, and that is within or close to the faculty's areas of expertise.

We have included a variety of sources, ranging from books to papers, videos, and podcasts. Most of these sources are accessible, and for the primary sources, we have provided abstracts or annotations.

In addition to application in a standard university program, these syllabi may also be useful for educators developing courses for industry practitioners. The structure may differ, but the content should still be valid for that audience as well.

This document is part of our transition plan for assisting educators who wish to implement a Master of Software Assurance degree program, specialization, or certificate program.

MSwA Implementation Considerations for Syllabi

Since there are a number of ways to order the courses, each university that wishes to implement an MSwA degree program or an SwA specialization within another degree program needs to consider topics that should appear in a course offered early on in its program. As well, there is a level of detail beyond the topics included in the syllabus that could be considered during implementation.

In order to provide a good background and understanding of the assurance problem, universities should expose students to a survey of attack patterns and vulnerabilities early in the curriculum. This will help them to "think like an attacker" and be mindful of the threat environment as they proceed through the program. For example, if the Assurance Assessment course appears early in the university's curriculum, it could include a discussion of attacks and vulnerabilities in the "security fundamentals" topic. The Common Weakness Enumeration (CWE) is a good resource to use in this discussion.

Students also need to understand the importance of social engineering. In recent attacks, such as the Google compromise in China[1], social engineering was used to get access to critical code sections. Most likely the Stuxnet attack[2] also involved aspects of social engineering. This topic could be included when instructors discuss the concept of "thinking like an attacker" in the Software Security Assurance course.

Hands-on topics that could be considered at a number of places in the curriculum include failure analysis, static analysis tools, and assurance testing. Detailed techniques such as fuzz testing may also be covered. These topics might appear in Assured Software Analytics or in one or more of the Assured Software Development courses. These courses are also good places to introduce common problems that appear at the architecture, design, and coding levels and result in vulnerabilities.

Since our field is relatively new, it is undoubtedly the case that recent research and industry trends will influence what is actually taught. Students should learn how to find the most up-to-date information from trustworthy places, such as the CWE website,[3] in order to stay current. A good exercise might be to ask students to review and rank software assurance websites.

---

1 For more information, see Google's blog at http://googleblog.blogspot.com/2010/01/new-approach-to-china.html.

2 More information about Stuxnet is available at http://en.wikipedia.org/wiki/Stuxnet.

3 http://cwe.mitre.org/

2 Assurance Management (AM) Course

2.1 Catalog Description

This course covers the fundamentals of software and system assurance management, including risk assessment, identification, analysis, mitigation, and monitoring for assurance; compliance with laws, regulations, standards, and policies related to assurance; planning and managing development projects that include assurance practices; and, given this information, making the business case for assurance.

2.2 Prerequisites

- completion of the Assured Software Development 1 (ASD1), Assured Software Development 2 (ASD2), and Assured Software Development 3 (ASD3), and Assurance Assessment (AA) courses. Alternatively, a code of practice, such as the Building Security In Maturity Model (BSIMM) [McGraw 2010], could be used as a source for practice selection (used in Weeks 5, 7, and 8) in lieu of more in-depth understanding of practices as conveyed in the ASD1, ASD2, and ASD3 courses. The AA course is strongly recommended as a prerequisite for the measurement topic that occurs in Week 9.
- Knowledge of project management in general and for software development in particular is helpful.

2.3 Expected Student Outcomes

After completing this course, students will be able to

1. understand basic risk management concepts
2. identify risks arising from vulnerabilities and threats
3. identify, analyze, plan for, mitigate, and monitor assurance risks
4. determine assurance processes and practices that mitigate risks
5. understand how to factor in compliance requirements (laws, regulations, standards, and policies) for assurance
6. understand how to add assurance considerations and practices as part of normal project management activities
7. identify, analyze, and select assurance practices that are relevant for a specific software development or acquisition project
8. make a business case for assurance

2.4 List of Topics

Topics on risk management concepts (Appendix A, Section 2.1) and process (Appendix A, Section 2.2) include

- risk types and classification, including different categories of risk such as business, project, and technical
- basic elements of risk analysis, including risk probability, impact, and severity
- models and processes used in risk management
- identification and classification of risks associated with a project
- analysis of the likelihood, impact, and severity of each identified risk
- risk management planning covering risk mitigation, avoidance, and acceptance
- the assessment and monitoring of the occurrence of risk and the management of risk mitigation strategies and actions

Topics on applying risk management concepts and process to software assurance (Appendix A, Section 2.3) include

- analyzing risks arising from threats and software flaws and vulnerabilities
- analyzing software assurance risks for new and existing systems
- planning for and mitigating software assurance risks
- identifying software assurance processes and practices that aid in mitigating and avoiding software assurance risks

Topics on compliance considerations for assurance (Appendix A, Section 4.3) include

- the extent to which selected laws and regulations are relevant for a specific software development or acquisition project, and how compliance might be demonstrated
- the extent to which selected standards are relevant for a specific software development or acquisition project, and how compliance might be demonstrated
- the development, deployment, and use of organizational policies to accelerate the adoption of software assurance practices, and how compliance might be demonstrated

Topics on managing assurance (Appendix A, Section 4.2) include

- extending normal software development (and acquisition) project management skills to include software assurance
- identifying, analyzing, and selecting software assurance practices that are relevant for a specific software development or acquisition project

Topics on making the business case for software assurance (Appendix A, Section 4.1) to develop and communicate cost-benefit arguments in support of deploying software assurance practices include

- financially based approaches, methods, models, and tools (such as valuation and cost-benefit models and cost and loss avoidance)

- risk analysis (as described above)
- business impact and needs analysis, specifically in support of business continuity and survivability

2.5 Sources

2.5.1 Primary

- Allen, Julia H.; Barnum, Sean; Ellison, Robert J.; McGraw, Gary; & Mead, Nancy R. *Software Security Engineering: A Guide for Project Managers*. Addison-Wesley Professional, 2008.

  Abstract from publisher

  Software that is developed from the beginning with security in mind will resist, tolerate, and recover from attacks more effectively than would otherwise be possible. While there may be no silver bullet for security, there are practices that project managers will find beneficial. With this management guide, you can select from a number of sound practices likely to increase the security and dependability of your software, both during its development and subsequently in its operation.

  Software Security Engineering draws extensively on the systematic approach developed for the Build Security In (BSI) Web site. Sponsored by the Department of Homeland Security Software Assurance Program, the BSI site offers a host of tools, guidelines, rules, principles, and other resources to help project managers address security issues in every phase of the software development life cycle (SDLC). The book's expert authors, themselves frequent contributors to the BSI site, represent two well-known resources in the security world: the CERT Program at the Software Engineering Institute (SEI) and Cigital, Inc., a consulting firm specializing in software security.

  This book will help you understand why

  - Software security is about more than just eliminating vulnerabilities and conducting penetration tests
  - Network security mechanisms and IT infrastructure security services do not sufficiently protect application software from security risks
  - Software security initiatives should follow a risk-management approach to identify priorities and to define what is "good enough"—understanding that software security risks will change throughout the SDLC
  - Project managers and software engineers need to learn to think like an attacker in order to address the range of functions that software should not do, and how software can better resist, tolerate, and recover when under attack

- Merkow, Mark S. & Raghavan, Lakshmikanth. *Secure and Resilient Software Development.* CRC Press, 2010.

  Abstract from publisher

  Although many software books highlight open problems in secure software development, few provide easily actionable, ground-level solutions. Breaking the mold, *Secure and Resilient Software Development* teaches you how to apply best practices and standards for consistent and secure software development. It details specific quality software development strategies and practices that stress resilience requirements with precise, actionable, and ground-level inputs.

  Providing comprehensive coverage, the book illustrates all phases of the secure software development life cycle. It shows developers how to master non-functional requirements including reliability, security, and resilience. The authors provide expert-level guidance through all phases of the process and supply many best practices, principles, testing practices, and design methodologies.

- Stoneburner, Gary; Hayden, Clark; & Feringa, Alexis. *Engineering Principles for Information Technology Security (A Baseline for Achieving Security).* National Institute of Standards and Technology (NIST), 2001.

2.5.2 Secondary

For risk management — preferred

- Alberts, Christopher J. & Dorofee, Audrey J. *Risk Management Framework* (CMU/SEI-2010-TR-071). Carnegie Mellon University, Software Engineering Institute, 2010. http://www.sei.cmu.edu/library/abstracts/reports/10tr017.cfm
- Australian/New Zealand Standard (AS/NZS) & International Organization for Standardization (ISO). AS/NZS ISO 31000: 2009 *Risk Management—Principles and Guidelines*, 1st ed. AS/NZS, November 2009.
- International Organization for Standardization (ISO). ISO/IEC FCD 27005: 2008 Information Technology—Security Techniques—Information Security Risk Management, 2nd ed. ISO, June 2008.

For risk management — backup if unable to purchase ISO standards

- Ross, Ron; Katzke, Stu; Johnson, Arnold; Swanson, Marianne; & Stoneburner, Gary. *Managing Risk from Information Systems: An Organizational Perspective* (NIST Special Publication 800-39), 2nd draft. National Institute of Standards and Technology, April 2008. http://www.smartgridinformation.info/pdf/2283_doc_1.pdf
- Joint Task Force Transformation Initiative. *Guide for Applying the Risk Management Framework to Federal Information Systems* (NIST Special Publication 800-37), Revision 1. National Institute of Standards and Technology, February 2010. http://csrc.nist.gov/publications/nistpubs/800-37-rev1/sp800-37-rev1-final.pdf

NIST SP 800-39 and NISP SP 800-37 apply risk management to U.S. federal agency information systems. Instructors and students should understand the key concepts and methods presented in these references but can safely ignore this specific application, generalizing to systems and software for all types of organizations.

- CMMI Product Team. *CMMI for Development, Version 1.3* (CMU/SEI-2010-TR-033). Carnegie Mellon University, Software Engineering Institute, November 2010. http://www.sei.cmu.edu/reports/10tr033.pdf

  See the RSKM process area on pages 349-361.

- CERT. *CERT Resilience Management Model*. http://www.cert.org/resilience/rmm.html (2010).

  See the Risk Management (RISK), Compliance (COMP), Resilient Technical Solution Engineering (RTSE) Process Areas. Free registration is required.

- International Organization for Standardization and International Electrotechnical Commission (ISO/IEC). *ISO/IEC 27002:2005 Information Technology – Security Techniques – Code of Practice for Information Security Management*. ISO/IEC, 2005.

  This document is also used extensively in System Operational Assurance. Purchase is required.

- Howard, Michael & Lipner, Steve. *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software*. Microsoft Press, 2006. An online version of the Microsoft SDL is available at http://www.microsoft.com/security/sdl/.

- Mansourov, Nicolai & Campara, Djenana. *System Assurance: Beyond Detecting Vulnerabilities*. Elsevier, 2011. http://www.elsevierdirect.com/ISBN/9780123814142/System-Assurance

  Abstract from Publisher

  In this day of frequent acquisitions and perpetual application integrations, systems are often an amalgamation of multiple programming languages and runtime platforms using new and legacy content. Systems of such mixed origins are increasingly vulnerable to defects and subversion. System Assurance: Beyond Detecting Vulnerabilities addresses these critical issues.

- McGraw, Gary; Chess, Brian; & Migues, Sammy. *Building Security In Maturity Model (BSIMM)*. http://bsimm.com/ (2010).

- Alberts, Christopher; Allen, Julia; & Stoddard, Robert. *Integrated Measurement and Analysis Framework for Software Security* (CMU/SEI-2010-TN-025). Software Engineering Institute, Carnegie Mellon University, 2010. http://www.sei.cmu.edu/library/abstracts/reports/10tn025.cfm

- Mead, Nancy R.; Allen, Julia H.; Conklin, W. Arthur; Drommi, Antonio; Harrison, John; Ingalsbe, Jeff; Rainey, James; & Shoemaker, Dan. *Making the Business Case for Software As-*

surance (CMU/SEI-2009-SR-001). Software Engineering Institute, Carnegie Mellon University, 2009. http://www.sei.cmu.edu/library/abstracts/reports/09sr001.cfm

2.6 Assignments

Students should complete individual assignments as described in the suggested schedule below. Assignments are discussed and assigned in the week shown and due the following week. Students should also work on a team project that includes developing a project plan and business case for a small software development project with software assurance requirements.

2.7 In-Class Activities

Class exercises should help students compare, analyze, and evaluate how organizations with successful software assurance initiatives (refer to the BSIMM) develop and present business case information, assess risk, select assurance practices, and plan their software assurance development projects. Additional in-class activities and demonstrations are described in the suggested schedule below. Note that depending on the time available, the number of activities could be increased or decreased.

2.8 Suggested Schedule

The syllabus in Table 1 below defines in-class discussions and other activities that are intended to reinforce lecture material and homework assignments.

Table 1: Syllabus for Assurance Management (AM) Course

Week	Topic	In-Class Activities	Suggested Readings	Assignment
1	**Introduction to Assurance Management** • Why security is a software issue • Managing for more secure software • Getting started • Preview of the entire course	• Discuss course objectives, content, and activities. • Discuss example software development project with software assurance requirements. It will be used throughout this course to demonstrate key concepts.	[Allen 2008] Chapters 1, 7, 8 [Mansourov 2011] Chapters 1, 2	

Week	Topic	In-Class Activities	Suggested Readings	Assignment
2	**Risk Management Concepts and Process** • Sources of risk • Define risk criteria based on business needs • Identify risks • Evaluate, categorize, and prioritize risks • Develop risk mitigation strategies • Implement mitigation actions • Review risks and adjust mitigation strategies	Apply a simplified version of the risk management process (identify, evaluate, mitigate, review) to a component of the example software development project. Include risk identification (using a simple high, medium, low categorization approach) and the determination of several mitigation approaches based on known software assurance practices.	• [Allen 2008] Chapter 7.4 • [Alberts 2010b] • [AS/NZS 2009] ISO 31000 (preferred) • [CERT 2010b] CERT-RMM RISK Process Area • [Ross 2008] NIST SP 800-39 (backup) • [Mansourov 2011] Chapter 5	Add details to the example presented in class for one or two specific risks.
3	**Applying Risk Management to Software Assurance** Tailor a general risk management process to software assurance risks during the software development life cycle.	Identify several typical risks by life-cycle phase	• [Alberts 2010] • [CMMI 2010] CMMI RSKM Process Area • [ISO 2008] ISO 27005 (preferred) • [Joint Task Force 2010] NIST SP 800-37 (backup)	Extend the list of risks developed in class. Propose possible mitigations to several identified risks.
4	**Compliance Considerations** • Laws and regulations • Policies • Standards	Demonstrate approaches for mapping software development project requirements and practices to the BSIMM, as one example of compliance with a “standard.”	• [CERT 2010b] CERT-RMM COMP Process Area • [McGraw 2010] BSIMM Compliance and Policy (CP) practice • [ISO 2008] ISO 27002 Section 15	Research and identify (or develop) an example of policy language that promotes the adoption of software assurance practices.

Week	Topic	In-Class Activities	Suggested Readings	Assignment
5	**Managing Software Assurance (Preparation)** • Review the SDLC and the integration of software assurance practices into the SDLC. (Refer to ASD1, 2, and 3.) • Present and discuss sample project; define a starter set of software assurance requirements.	Discussion of sample project and how to begin analyzing it. Consider whether to break into teams to do subsequent work or perform work at the individual student level.	• [Allen 2008] Chapters 3, 4, 5 • [Mansourov 2011] Chapters 3, 4 • [Merkow 2010] Chapters 2, 3, 4	Elaborate on software assurance requirements, identify and add example compliance requirements (Week 4), and derive additional software assurance requirements. This gives students/teams an opportunity to tailor the software project to their specific interests.
6	**Managing Software Assurance (Identify Risks)** Demonstrate the process for identifying risks building upon Weeks 2 and 3, applied to the sample project	In-class teams discuss and report risks for sample project.	Weeks 2, 3 readings	Identify risks for sample project.
7	**Managing Software Assurance (Select Practices)** Demonstrate the process for selecting practices to mitigate risks building upon Week 3, applied to the sample project	In-class teams discuss and report practices to mitigate risks for sample project.	• Week 3 readings • [Allen 2008] Chapter 8 • [CERT 2010b] CERT-RMM RTSE Process Area • [McGraw 2010] BSIMM • [Howard 2006] Microsoft SDL • [Merkow 2010] Chapters 5, 6, 8 • [ISO 2008] ISO 27002 Sections 12.1-12.5	Identify practices to mitigate selected risks for sample project.
8	**Managing Software Assurance (Plan Development)** Plan the sample project: demonstrate plan components (tasks, resources, schedule)	In-class teams discuss tasks, resources, and schedule for sample project.	• [Allen 2008] Chapter 7.5 • [McGraw 2010] BSIMM Strategy and Metrics • [Howard 2006] Microsoft SDL, Appendix O	Identify tasks, resources, and schedule for sample project.

Week	Topic	In-Class Activities	Suggested Readings	Assignment
9	**Managing Software Assurance (Review and Measure)** • Performing status reviews of sample project • Measuring key indicators of sample project status • Revisiting risks as project progresses	In-class teams discuss review criteria, key indicators, and triggers for revising risks and mitigations for sample project.	[Alberts 2010b] particularly Appendix A	• Define review criteria and key indicators for sample project. • Identify several review or indicator measures that may trigger new or revised risks. • Prepare for in-classroom reviews.
10, 11	**Managing Software Assurance (Recap and Lessons Learned)** • Define classroom review criteria • Conduct a "mock" review including key indicators and revised risks • Capture lessons learned	Selected individuals or teams present a review of their sample project. Each presentation is "scored" by student reviewers.		• Prepare for in-classroom reviews. • Hand in an annotated presentation (slides plus notes). • Student reviewers hand in scores and rationale.
12	**Making the Business Case (Methods)** Review business case methods	Apply one method to sample project.	[Mead 2009]	Apply two additional methods to sample project. Prepare presentation.
13	**Making the Business Case (Application)** Present business case for sample project	Selected presentations and critique	[Mead 2009]	Prepare for final exam.
14	**Final Exam**			

3 System Operational Assurance (SOpA) Course

3.1 Catalog Description

This course covers how to establish procedures to assure that systems in operation continue to meet their security requirements and can respond to new threats. Students will learn about assurance policies and procedures; assurance training; technologies for monitoring and controlling systems; evaluation of monitoring results; maintenance of operational systems; evaluation of malicious code; responding to adverse events; and the actions necessary for maintaining business survivability and continuity of operations.

3.2 Prerequisites

Knowledge of

- the software development life cycle (gained through an undergraduate software engineering course)
- security issues (gained through an undergraduate Introduction to Security course, work experience, or remedial work)

3.3 Expected Student Outcomes

After completing this course, students will be able to

1. understand the role of business objectives and strategic planning in software and system assurance
2. create appropriate security policies and procedures for system operations
3. understand the type of training needed by users and administrative personnel in secure system operations
4. understand the capabilities and limitations of monitoring technologies for systems, services, and personnel
5. evaluate operational monitoring results for system and service functionality and security
6. maintain and evolve operational systems while preserving assured functionality and security
7. evaluate malicious content and apply appropriate countermeasures
8. plan for and execute effective responses to operational system accidents, failures, and intrusions
9. maintain business survivability and continuity of operations in adverse environments

3.4 List of Topics

Topics on operational procedures (Appendix A, Section 7.1) include

- the role of business objectives and strategic planning in assuring that operational systems continue to function as intended
- the development of security policies and procedures for secure system operations
- the selection of training for users and system administrative personnel in secure system operations

Topics on operational monitoring (Appendix A, Section 7.2) include

- the capabilities and limitations of monitoring technologies and the installation and configuration/acquisition of monitors and controls for systems, services, and personnel
- the evaluation of operational monitoring results with respect to system and service functionality and security
- the maintenance and evolution of operational systems while preserving assured functionality and security. This includes understanding new threats and the countermeasures for addressing them.
- the evaluation of malicious content and applying countermeasures to mitigate the risks and contain the damage caused by such content

Topics on system control (Appendix A, Section 7.3) include

- control of operational systems, including planning for and executing effective responses to operational system accidents, failures, and intrusions
- business survivability and continuity of operations in adverse environments

3.5 Sources

3.5.1 Primary

- Stoneburner, Gary; Hayden, Clark; & Feringa, Alexis. *Engineering Principles for Information Technology Security (A Baseline for Achieving Security).* National Institute of Standards and Technology (NIST), 2001.

Preferred

- International Organization for Standardization and International Electrotechnical Commission (ISO/IEC). *ISO/IEC 27002:2005 Information Technology—Security Techniques—Code of Practice for Information Security Management*. ISO/IEC, 2005.

  This document is also used in Assurance Management. Purchase is required.

Backup if unable to purchase ISO standards

- Joint Task Force Transformation Initiative. *Recommended Security Controls for Federal Information Systems and Organizations* (NIST Special Publication 800-53), Revision 3. National Institute of Standards and Technology, August 2009. Updated May 2010. http://csrc.nist.gov/publications/nistpubs/800-53-Rev3/sp800-53-rev3-final_updated-errata_05-01-2010.pdf

3.5.2 Secondary

- Caralli, Richard; Stevens, James F.; Bradford, J. Wilke; & Wilson, William R. *The Critical Success Factor Method: Establishing a Foundation for Enterprise Security Management* (CMU/SEI-2004-TR-010). Carnegie Mellon University, Software Engineering Institute, July 2004. http://www.sei.cmu.edu/library/abstracts/reports/04tr010.cfm
- The SANS Institute. *Introduction to the SANS Security Policy Project*. http://www.sans.org/security-resources/policies/ (2011).
- CERT. *CERT Resilience Management Model*. http://www.cert.org/resilience/rmm.html (2010).

  See the Monitoring (MON), Organizational Training and Awareness (OTA), Incident Management (IMC), Vulnerability Analysis and Resolution (VAR), and Service Continuity (SC) Process Areas. Free registration is required.

- Howard, Michael & Lipner, Steve. *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software*. Microsoft Press, 2006. An online version of the Microsoft SDL is available at http://www.microsoft.com/security/sdl/.

  Phase Five: Release is most applicable. Ignore discussions on privacy.

- McGraw, Gary; Chess, Brian; & Migues, Sammy. *Building Security In Maturity Model (BSIMM)*. http://bsimm.com/ (2010).

  The Deployment practice is most applicable.

- Mell, Peter; Kent, Karen; & Nusbaum, Joseph. *Guide to Malware Incident Prevention and Handling* (NIST Special Publication 800-83). National Institute of Standards and Technology, November 2005. http://csrc.nist.gov/publications/nistpubs/800-83/SP800-83.pdf
- Scarfone, Karen; Grance, Tim; & Masone, Kelly. *Computer Security Incident Handling Guide* (NIST Special Publication 800-61), Revision 1. National Institute of Standards and Technology, March 2008. http://csrc.nist.gov/publications/nistpubs/800-61-rev1/SP800-61rev1.pdf
- Swanson, Marianne; Bowen, Pauline; Phillips, Amy Wohl; Gallup, Dean; & Lynes, David. *Contingency Planning Guide for Federal Information Systems* (NIST Special Publication 800-34), Revision 1. National Institute of Standards and Technology, May 2010. http://csrc.nist.gov/publications/nistpubs/800-34-rev1/sp800-34-rev1_errata-Nov11-2010.pdf

3.6 Assignments

Students should complete individual assignments as described in the suggested schedule. Assignments are discussed and assigned in the week shown and due the following week. Students should also work on a team project that includes developing sample artifacts (policies and procedures, training, monitoring and control approaches, etc.) for a software application with software assurance requirements that is about to be deployed or is in operations/production.

3.7 In-Class Activities

In-class activities and demonstrations are described in the suggested schedule below. Note that depending on the time available, the number of activities could be increased or decreased.

3.8 Suggested Schedule

The syllabus in Table 2 below defines in-class discussions and other activities that are intended to reinforce lecture material and homework assignments.

Table 2: Syllabus for the System Operational Assurance (SOpA) Course

Week	Topic	In-Class Activities	Suggested Readings	Assignment
1	**Introduction to System Operational Assurance** • Operational policies and procedures • Operational monitoring • System control	• Discussion of course objectives, content, and activities • Introduce and discuss selected software application with software assurance requirements that is about to be deployed or is in operations/ production. This example will be used to illustrate key learning points throughout the course.	• [Caralli 2004] Chapter 4 • [ISO 2005] ISO 27002 Sections 6, 10 • [Joint Task Force 2009] NIST 800-53 Chapters 2, 3	
2	**Policies and Procedures for Secure System Operations** • Define policy, standard, guideline, procedure • Policy life cycle (scope, implementation, enforcement, review, revision) • Policy and procedures for secure system operations	Walk through an example policy and one supporting procedure for the example software application. Provide a template for the assignment.	• [McGraw 2010] BSIMM Compliance and Policy (CP) practice • [ISO 2005] ISO 27002 Section 5, 10.1 • [Joint Task Force 2009] NIST 800-53 Appendix F, specifically the policy and procedures control (-1) in each of the 18 security control classes • [SANS 2011]	Search for and identify several examples of reasonable operational security policies related to software assurance and software security. Place in template form.

Week	Topic	In-Class Activities	Suggested Readings	Assignment
3	**Training** • Awareness and training topics for secure system operations • Finding relevant training • Defining an awareness and training program	Sketch out an example awareness and training program for the example software application.	• [CERT 2010b] CERT-RMM OTA • [McGraw 2010] BSIMM Training (T) practice • [ISO 2008] ISO 27002 Section 8.2.2 • [Joint Task Force 2009] NIST 800-53 Appendix F, AT	Add details and populate the in-class defined program with relevant topics and sources.
4	**Operational Monitoring (Program and Tools)** • Defining a monitoring program and monitoring requirements for software, systems, and personnel. Discuss roles and responsibilities for those executing the program. • Identifying monitoring tools, techniques, and methods including alert services	Define a starter monitoring program for the example software application.	• [CERT 2010b] CERT-RMM MON • [ISO 2008] ISO 27002 Section 10.10 • [McGraw 2010] BSIMM Penetration Testing (PT) • Also search for and review incident advisory and alert services such as those provided by US-CERT. http://www.us-cert.gov	Search for and review open source software monitoring tools; establish criteria for selecting and then select the tools that fit best with the example software application.
5	**Operational Monitoring (Information Collection and Reporting)** • Collecting and recording monitoring information • Reporting monitoring results • Evaluating monitoring results, taking action where required	Based on several selected tools, discuss how they collect, record, and report monitoring results.	• Week 4 readings • [ISO 2008] ISO 27002 Section 12.4 • [Joint Task Force 2009] NIST 800-53 Appendix F (search on “monitor” for applicable controls)	Select one open source tool and develop a short white paper on how it collects, records, and reports information for a software application of the student’s choosing.

Week	Topic	In-Class Activities	Suggested Readings	Assignment
6	**Maintaining Operational Systems (Managing the Environment)** • Managing the operational software environment • Configuration and change management • Vulnerability management	• Explore Center for Internet Security resources. • Explore Common Vulnerability and Exposures resources. http://cve.mitre.org/ • Discuss how to use these in an operational setting.	• [ISO 2008] ISO 27002 Sections 12.4, 12.5, 12.6 • [Joint Task Force 2009] NIST 800-53 Appendix F, CM, RA (vulnerability scanning) • [McGraw 2010] BSIMM Software Environment (SE) practice • [McGraw 2010] BSIMM Configuration Management and Vulnerability Management (CMVM) practice • [CERT 2010b] CERT-RMM VAR • Center for Internet Security configuration benchmarks http://cisecurity.org/en-us/	Write a short white paper that describes how vulnerabilities discovered during operations could have been addressed earlier in the software development life cycle. In addition, for the example software application, select configuration benchmarks and settings.
7	**Evaluating Malicious Content** • Malware categories • Malware incident prevention • Malware incident response (covered more generally for all types of security incidents during Weeks 8 and 9, so may want to defer this topic)	Possible demonstration of several malware incidents such as those described in NIST 800-53 Appendix B	• [Mell 2005] NIST 800-83 • [Scarfone 2008] NIST 800-61 Section 5 • [ISO 2008] ISO 27002 Section 10.4 • [Joint Task Force 2009] NIST 800-53 Appendix F, SI	Write a short white paper that describes how security technologies, such as those listed in NIST 800-83 Appendix A, could be deployed to protect the example software application in its production environment. Take cost, resources, schedule, and technology priorities into account.

Week	Topic	In-Class Activities	Suggested Readings	Assignment
8	**Maintaining Operational Systems (Security Incident Management 1)** • Plan for incident management. • Detect and analyze incidents.	• Walk through the detection and analysis of several security incidents. • Discuss various team structures for incident response.	• [Scarfone 2008] NIST 800-61 • [CERT 2010b] CERT-RMM IMC • [ISO 2008] ISO 27002 Section 13 • [Joint Task Force 2009] NIST 800-53 Appendix F, IR • [Killcrece 2008] https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/incident/223-BSI.html	Each course team prepares a short presentation that describes the full life cycle of an incident against the example software application, through recovery. Do not include postmortem review and lessons learned in the classroom presentation, but capture this and submit it to the course instructor.
9	**Maintaining Operational Systems (Security Incident Management 2)** • Respond to and recover from incidents • Incident postmortem review and lessons learned	• Walk through the response to and recovery from several security incidents. • Conduct an incident postmortem review and identify lessons learned.	Same as Week 8	(Same as Week 8) Each course team prepares a short presentation that describes the full life cycle of an incident against the example software application, through recovery. Do not include postmortem review and lessons learned in the presentation, but capture this and submit it to the course instructor.
10	**Incident Presentations and Discussion** • Team presentations	Team presentations	Same as Week 8	Write a short white paper identifying lessons learned and improvement actions that should be taken as a follow-up to each presented incident.
11	**Business Continuity Planning** • Planning including business impact analysis • Business impact analysis • Contingency planning and the SDLC (NIST 800-34 Appendix F)	Conduct an example business impact analysis for a facility, a system, and key personnel.	• [CERT 2010b] CERT-RMM SC • [Swanson 2010] NIST 800-34 • [ISO 2008] ISO 27002 Section 14 • [Joint Task Force 2009] NIST 800-53 Appendix F, CP	Write a short white paper of a business impact analysis for the example software application.

Week	Topic	In-Class Activities	Suggested Readings	Assignment
12	**Business Continuity Exercise and Test** • Exercise and test • Measure plan effectiveness	• Conduct a table-top exercise based on a disruption of service continuity. • Define review criteria for Week 13 presentations.	Same as Week 11	Each course team prepares a short presentation/ demonstration of a table-top exercise for the example software application.
13	**Business Continuity Presentations**	Team presentations. Those not presenting complete evaluations using criteria defined in Week 12 class.	Same as Week 11	Prepare for final exam.
14	**Final Exam**			

4 Assured Software Analytics (ASA) Course

4.1 Catalog Description

This course covers analysis methods, techniques, and tools to help assure that newly developed and acquired software, systems, and services meet their functional and security requirements. Students will learn methods for structural and functional analysis of software components "in the small" and analysis of software systems "in the large." They will also learn concepts of testing for assurance and developing auditable assurance evidence.

4.2 Prerequisites

- Assured Software Development 1 (ASD1) course

4.3 Corequisites

- Assured Software Development 2 (ASD2) course

4.4 Expected Student Outcomes

After completing this course, students will be able to

1. understand methods for assurance of system networks, architectures, and components
2. apply structuring methods to components to improve understandability and modifiability
3. apply reverse engineering to components to assess functionality and security properties
4. assess system architectures and develop and apply assurance plans
5. understand capabilities and limitations of methods, techniques, and tools for software analysis
6. evaluate testing and inspection methods, plans, and results for assuring software
7. apply methods for assuring acquired software and services
8. understand requirements for auditable assurance evidence

4.5 List of Topics

Topics on assurance analytics for new and existing software (Appendix A, Sections 6.2 and 6.3) include

- analysis of assurance properties of system networks, architectures, and components
- application of structuring techniques to systematize the logic of existing software
- application of reverse engineering techniques to reveal functionality and security properties of existing software
- capabilities and limitations of methods and tools for software analysis
- assessment of assurance capabilities and evidence produced by testing and inspections

- requirements for auditable assurance evidence

Topics on assurance analytics for acquisition (Appendix A, Section 6.4) include

- assurance of software acquired through supply chains, vendors, and open source
- assurance of acquired service functionality and security

4.6 Sources

4.6.1 Primary

- Linger, R.; Mills, H.; & Witt, B. *Structured Programming: Theory and Practice.* Addison-Wesley, 1979.

  Sessions 1-4 of this course provides an exposure to rigorous methods for assurance analysis "in the small," with applicability to particular system components identified as requiring detailed investigation. The functional approach to software analysis is well suited to this objective. This textbook defines foundations for structuring and reverse engineering of software to verify functionality. The book is out of print, but used copies can be obtained, and it will likely be available online.

- Committee for Advancing Software-Intensive Systems Producibility; Computer Science and Telecommunications Board; & Division on Engineering and Physical Sciences. *Critical Code: Software Producibility for Defense*. National Academy of Sciences, 2010.

  This volume describes "in the large" software assurance issues and solution strategies, chiefly from a DoD perspective but with substantial applicability to commercial sectors.

4.6.2 Secondary

Given the pace of technology development, coursework in software assurance is, of necessity, a moving target. The following references are suggestive of what is currently available. Additional or different materials can be considered at the time of course delivery.

- Wysopal, Chris; Nelson, Lucas; Dai Zovi, Dino; & Dustin, Elfriede. *The Art of Software Security Testing*: *Identifying Software Security Flaws.* Addison-Wesley Professional, 2006.

  This book reviews design and coding vulnerabilities that can arise in software, providing guidance for avoiding them. It discusses customization of debugging tools to test the unique aspects of programs and analyze the results to identify exploitable vulnerabilities.

- Eagle, Chris. The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler. No Starch Press, 2008.

  This textbook describes a widely used tool that supports automated analysis of software executables, with the principal focus on malware.

- Holt, Alan & Huang, Chi-Yu, 802.11 Wireless Networks: Security and Analysis. Springer, 2010.

This textbook includes wireless network security analysis and methods. It provides a reference for assurance issues in wireless networks that support software and service operations across organizations and that are themselves software enabled.

- Department of Homeland Security (DHS) Software Assurance (SwA) Acquisition Working Group. *Software Assurance in Acquisition: Mitigating Risks to the Enterprise*. https://buildsecurityin.us-cert.gov/swa/downloads/SwA_in_Acquisition_102208.pdf (2008).

  This document provides a comprehensive view of assurance issues and procedures in software acquisition. It is compiled from a government perspective but is relevant to private sector acquisition as well.

- Epstein, Jeremy; Matsumoto, Scott; & McGraw. "Software Security and SOA: Danger, Will Robinson!" *IEEE Security & Practice 4*, 1 (January/February 2006): 80–83.

  This paper highlights security assurance issues in service-oriented architectures.

- IBM. *IBM Point of View: Security and Cloud Computing.* http://www.ibm.com/common/ssi/fcgi-bin/ssialias?infotype=SA&subtype=WH&appname=SWGE_TI_SE_USEN&htmlfid=TIW14045USEN&attachment=TIW14045USEN_HR.PDF (2009).

  This paper discusses security issues in cloud computing environments.

- Thiagarajan, Val. *Information Security Management: BS 7799.2:2002: Audit Check List for SANS*. http://www.sans.org/score/checklists/ISO_17799_checklist.pdf (2003).

  This checklist covers many aspects of assurance auditing.

- National Institute of Standards and Technology (NIST). *SAMATE—Software Assurance Metrics and Tool Evaluation*. http://samate.nist.gov/Main_Page.html (2005).

  This project is dedicated to the identification, testing, and measurement of tools for a variety of purposes.

- Walton, G, Linger, R., and Longstaff, T. "Computational Evaluation of Software Security Attributes," 1–10. *Proceedings of the 42nd Hawaii International Conference on System Sciences*. Los Alamitos, CA, Jan. 2009. IEEE Computer Society Press, 2009.

  This paper describes security attribute assurance in terms of implementations and how to check them. While the paper discusses methods for computational evaluation of implementations, the manual methods described in the course can be effective as well.

4.7 Assignments

The suggested readings and homework assignments at the time of course offering should reflect technology evolution and topic emphasis over time. Representative team assignments are suggested in the syllabus below. Teams can be defined for the duration of the course, typically with four to six members, and can select a leader for each assignment. Reports to the class can be graded and should be short, yet comprehensive at an appropriate level of abstraction. This course is also ideal for longer duration, team-based case studies selected by the instructor. Assignments are discussed and assigned in the week shown and due the following week.

4.8 In-Class Activities

In-class activities and demonstrations are described in the suggested schedule below. Note that depending on the time available, the number of activities could be increased or decreased.

4.9 Suggested Schedule

The syllabus in Table 3 below defines in-class discussions and other activities that are intended to reinforce lecture material and homework assignments. Guest speakers with real-world experience in particular course topics should also be considered.

Table 3: Syllabus for the Assured Software Analytics (ASA) Course

Week	Topic	In-Class Activities	Suggested Readings	Assignment
1	**Introduction** • Course structure and objectives; consequences of exploitations and failures for various types of systems; goals of assurance analytics • Overview of assurance analytics for new and legacy applications and networks, and for acquired software and services • Roadmap for Weeks 2-5 as a brief exposure to rigorous software analysis of sequential logic for assurance “in the small” • Foundations I: Programs as implementations of mathematical functions; fundamental control structures; design language forms	• Discuss what it means to have “software-defined” products, such as aircraft avionics, and businesses, such as banking or online retailing. • Discuss the consequences of exploitations and failures for various systems, such as banking, power generation, telecommunications, aircraft avionics, video games, and social networks. • Discuss and define imagined assurance requirements for a university information system, from business and technical perspectives.	[Linger 1979] Chapter 3	

Week	Topic	In-Class Activities	Suggested Readings	Assignment
2	**Foundations II: Analysis of Program Structure** • The Structure Theorem and its constructive proof • Application of the constructive proof to program structuring • Comparison of unstructured and structured logic	• Selected team reports and discussion • Apply the Structure Theorem proof to transform the logic of a small unstructured program expressed in design language form.	[Linger 1979] Chapter 4	Team application of the Structure Theorem proof to structuring a given spaghetti-logic program expressed in design language form
3	**Foundations III: Analysis of Program Functionality** • The algebraic structure of structured programs as a basis for reverse engineering • Reverse engineering as a documentation process for control structures • Reverse engineering of control structure functionality through trace table analysis • Data structures for verification in design language form	• Selected team reports and discussion • Apply reverse engineering techniques to a small program expressed in design language form	[Linger 1979] Chapter 5	Team reverse engineering of the function of a given structured program expressed in design language form
4	**Foundations IV: Verification of Program Functionality** • The Correctness Theorem • Application of trace table analysis to verification of program functionality	• Selected individual reports and discussion • Verify the functionality of a small program in design language form	[Linger 1979] Chapter 6	Individual verification of the function of a given structured program in design language form
5	**Assurance of Software Systems** • Processes for scaling up to "assurance in the large" • System environment, architecture, component, interaction, and dependency discovery • Assessing system requirements, specifications, designs, and history of development and evolution • Architecture properties including modularity, resource sharing, quality, documentation, and traceability • Assessing information loss from poor documentation, complexity, and lack of traceability • Complexity reduction and quality improvement	Selected team reports and discussion of findings	[Committee 2010] Chapters 2-4	Team investigation of a given architecture for a real system to assess architectural properties, quality, and information availability, and develop a plan for assurance that includes both "in the large" and "in the small" aspects

Week	Topic	In-Class Activities	Suggested Readings	Assignment
6	**Assurance of Security Properties** • Assuring security properties, such as authentication, authorization, non-repudiation, confidentiality, privacy, integrity, and availability • Assuring encryption methods used in security implementations	Selected individual reports and discussion	[Walton 2009]	Individual web-search determination of best practices in assuring security properties
7	**Methods and Tools I: Application Level** • Survey of capabilities and limitations of analysis tools for evaluating and improving functionality and security at the application level, with emphasis on detection of vulnerabilities • Introduction to specific popular tools for static and dynamic analysis • Special topic: overview of Ida Pro tool for malware analysis	Selected team reports and discussion	[NIST 2005] Selected readings on tools discussed in the session	Team evaluation of assigned tool capabilities and limitations for supporting software assurance at the application level, for both static and dynamic analysis
8	**Methods and Tools II: Network Level** • Survey of capabilities and limitations of analysis tools for evaluating and improving functionality and security at the network level, with emphasis on network analysis, and monitoring through intrusion and anomaly detection methods • Introduction to specific popular tools for network analysis and monitoring	Selected team reports and discussion	Selected readings on tools discussed in the session	Team evaluation of assigned tool capabilities and limitations for supporting software assurance at the network level, for both static and dynamic analysis
9	**Assurance Testing** • Evaluating test and inspection plans from an assurance perspective • Analyzing vulnerability detection capabilities of test and inspection methods • Analyzing threat environment coverage of penetration tests • Assessing software function and security based on test and inspection results	Selected team reports and discussion	[Wysopal 2006] Chapters 4, 5	Team development of a penetration test plan for an assigned organization, for example, an online retailer, including an assessment of the threat environment and expected assurance evidence from the testing

Week	Topic	In-Class Activities	Suggested Readings	Assignment
10	**Assuring Acquired Software** • Sources, properties, benefits, and risks of acquired software, including vendor supply chains, open source, and COTS • Evaluating supplier documentation and assurance claims • Assurance methods for acquired software, including testing and code analysis	• Selected team reports and discussion • Assurance analysis of a downloaded COTS program	[DHS 2008]	Team web-search determination of best practices for assuring functionality and security of acquired software
11	**Assuring Acquired Services** • Software service properties, requirements, and delivery, including SOA and cloud computing • Service properties including scalability, maintainability, reliability, availability, performance, and security • Service and network provider assessment, Service Level Agreement definition, and service metrics and monitoring	• Selected team reports • Define key elements of an example Service Level Agreement with a cloud computing vendor, such as for university network services • Create a plan for ongoing assurance of service delivery, including metrics.	[IBM 2009] websites that track reliability of cloud computing services ([CloudFail 2011] is one.)	Team web-search determination of top 10 best practices for assuring functionality and security of acquired services
12	**Assurance Evidence** • Business, legal, and regulatory requirements for assurance evidence • What does and does not constitute auditable evidence • Assurance auditing processes, including checklist–based and tool-augmented approaches • Assurance auditing for SOA and cloud computing environments	• Team reports • Discussion of value and shortcomings of checklist-based and tool-based auditing processes. • Discussion of audit evidence in cloud computing. • Discussion of embedding assurance auditing into business processes.	[Thiagarajan 2003]	Team web-search investigation of legal and regulatory requirements for assurance evidence and auditing, plus best practices and checklists
13	**Assurance for Human Factors, and Course Review** • Assurance aspects of human elements of systems • Sources of human errors and malicious intent that impact security, including insider aspects, and how to address them • Assessing user training and monitoring for human-computer interaction • Course review	• Selected team reports and discussion • Discussion of assurance for human elements of systems, including preventative and corrective actions for errors and malicious behavior		• Team web-search investigation of human errors and malicious intent that have caused security problems, and means to avoid them • Individual questions for course review
14	**Final Exam**			

5 Assured Software Development 1 (ASD1) Course

5.1 Catalog Description

This course covers the fundamentals of incorporating assurance practices, methods, and technologies into software development and acquisition life-cycle processes and models. With this foundation, the course provides students with rigorous methods for eliciting software and system assurance requirements; using threat identification, characterization, and modeling; assurance risk assessment; and misuse/abuse cases. Students will also learn how to evaluate methods and environments for creating software and systems that meet their functionality and security requirements.

5.2 Prerequisites

Knowledge of

- the software development life cycle (SDLC) and its activities (gained through an undergraduate software engineering course, software development work experience, or remedial work)
- security issues (gained through an undergraduate Introduction to Security course, work experience, or remedial work)

5.3 Expected Student Outcomes

After completing this course, students will be able to

1. understand life-cycle models and processes for newly developed software systems
2. understand life-cycle models and processes for the acquisition, supply, and service of a software system
3. use methods, techniques, and tools to assess the applicability of assurance processes and practices for typical life-cycle phases, such as requirements engineering, architecture and design, coding, test, evolution, acquisition, and retirement
4. elicit and analyze requirements for assured software, based on threat modeling, identification of attack patterns, and misuse/abuse cases
5. apply security requirements engineering methods in developing assurance requirements

5.4 List of Topics

Topics on new development of software life-cycle processes (Appendix A, Section 1.1.1) and integration, assembly, and deployment of those processes (Appendix A, Section 1.1.2) include

- the software process: understanding life-cycle processes associated with full development of a new software system. This includes a general understanding of process models such as iterative development, spiral model, waterfall, and agile, as well as life-cycle activities.

Topics on the operation and evolution of software life-cycle processes (Appendix A, Section 1.1.3) include

- understanding processes that guide the operation of the software system and its evolution over time

Topics on the acquisition, supply, and service of software life-cycle processes (Appendix A, Section 1.1.4) include

- understanding processes that support the acquisition, supply, or service of a software system. This is where processes such as Common Criteria could be taught.

Topics on the process and practice assessment of software assurance processes and practices (Appendix A, Section 1.2.1) include

- the software assurance process: learning and applying methods, procedures, and tools used to assess assurance processes and practices. This is where complete life-cycle processes such as CLASP could be taught and best practices models such as the BSIMM, SAFECode, and OWASP could be taught.

Topics on the software assurance integration into SDLC phases (Appendix A, Section 1.2.2) include

- the software assurance process: learning how to integrate and apply assurance practices into typical life-cycle phases, such as requirements engineering, architecture and design, coding, test, evolution, acquisition, and retirement. The focus here is specifically on practices that improve assurance. This topic could include Microsoft SDL and activities that apply to the early life-cycle phases, such as threat modeling, assurance risk assessment,[4] attack trees, and misuse/abuse cases.

Topics on the software assurance integration into SDLC phases (Appendix A, Section 1.2.2) include

- how to evaluate the capabilities and limitations of technical environments, languages, and tools with respect to creating assured software functionality and security. Of particular interest are environments that support assurance, languages that provide fewer opportunities to insert vulnerabilities, and tools used to improve assurance at various phases in the life cycle.

Topics on improvement of assurance technology (Appendix A, Section 6.1.2) include

- how to assess and recommend improvements in assurance methods as needed within project constraints, including cost, schedule, functionality, and quality factors

[4] Note that risk assessment is covered to the extent that it is needed for requirements engineering. The risk management process is fully covered in the Assurance Management course.

Topics on the assured software development methods (Appendix A, Sections 6.2 and 6.2.1 [requirements]) include

- rigorous methods for developing assured system and software requirements and specifications, and how to apply those methods. This includes requirements engineering processes that are specific to assured systems, as well as risk analysis, requirements elicitation, and prioritization methods. As part of these methods, students will need to apply techniques such as threat modeling, attack trees, and misuse/abuse cases. This topic also includes determining whether the requirements are feasible and inspecting them.

5.5 Sources

5.5.1 Primary

- Merkow, Mark S. & Raghavan, Lakshmikanth. *Secure and Resilient Software Development.* CRC Press, 2010.

  Abstract from publisher

  Although many software books highlight open problems in secure software development, few provide easily actionable, ground-level solutions. Breaking the mold, *Secure and Resilient Software Development* teaches you how to apply best practices and standards for consistent and secure software development. It details specific quality software development strategies and practices that stress resilience requirements with precise, actionable, and ground-level inputs.

  Providing comprehensive coverage, the book illustrates all phases of the secure software development life cycle. It shows developers how to master non-functional requirements including reliability, security, and resilience. The authors provide expert-level guidance through all phases of the process and supply many best practices, principles, testing practices, and design methodologies.

- Allen, Julia H.; Barnum, Sean; Ellison, Robert J.; McGraw, Gary; & Mead, Nancy R. *Software Security Engineering: A Guide for Project Managers.* Addison-Wesley Professional, 2008.

  Abstract from publisher

  Software that is developed from the beginning with security in mind will resist, tolerate, and recover from attacks more effectively than would otherwise be possible. While there may be no silver bullet for security, there are practices that project managers will find beneficial. With this management guide, you can select from a number of sound practices likely to increase the security and dependability of your software, both during its development and subsequently in its operation.

  Software Security Engineering draws extensively on the systematic approach developed for the Build Security In (BSI) Web site. Sponsored by the Department of Homeland Security Software Assurance Program, the BSI site offers a host of tools, guidelines, rules, principles, and other resources to help project managers address security issues in every phase of the software development life cycle (SDLC). The book's expert authors, themselves frequent

contributors to the BSI site, represent two well-known resources in the security world: the CERT Program at the Software Engineering Institute (SEI) and Cigital, Inc., a consulting firm specializing in software security.

This book will help you understand why

- Software security is about more than just eliminating vulnerabilities and conducting penetration tests
- Network security mechanisms and IT infrastructure security services do not sufficiently protect application software from security risks
- Software security initiatives should follow a risk-management approach to identify priorities and to define what is "good enough"—understanding that software security risks will change throughout the SDLC
- Project managers and software engineers need to learn to think like an attacker in order to address the range of functions that software should not do, and how software can better resist, tolerate, and recover when under attack

- Howard, Michael & Lipner, Steve. The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software. Microsoft Press, 2006. An online version of the Microsoft SDL is available at http://www.microsoft.com/security/sdl/.

  Abstract from publisher

  Your in-depth, expert guide to the proven process that helps reduce security bugs. Your customers demand and deserve better security and privacy in their software. This book is the first to detail a rigorous, proven methodology that measurably minimizes security bugs—the Security Development Lifecycle (SDL). In this long-awaited book, security experts Michael Howard and Steve Lipner from the Microsoft Security Engineering Team guide you through each stage of the SDL—from education and design to testing and post-release. You get their first-hand insights, best practices, a practical history of the SDL, and lessons to help you implement the SDL in any development organization.

  Discover how to:

  - Use a streamlined risk-analysis process to find security design issues before code is committed
  - Apply secure-coding best practices and a proven testing process
  - Conduct a final security review before a product ships
  - Arm customers with prescriptive guidance to configure and deploy your product more securely
  - Establish a plan to respond to new security vulnerabilities
  - Integrate security discipline into agile methods and processes, such as Extreme
  - Programming and Scrum

  Includes a CD featuring:

  - A six-part security class video conducted by the authors and other Microsoft security experts

- Sample SDL documents and fuzz testing tool

5.5.2 Secondary

- Ahern, Dennis M.; Clouse, Aaron; & Turner, Richard. *CMMI Distilled: A Practical Introduction to Integrated Process Improvement*. 3rd ed. Addison-Wesley Professional, 2008.
- Garcia, Suzanne & Turner, Richard. *CMMI Survival Guide: Just Enough Process Improvement*. Addison-Wesley Professional, 2006.
- Department of Homeland Security (DHS). *Security Requirements Engineering* (articles). https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/requirements.html (2010).
- Department of Homeland Security (DHS). *Build Security In, Secure Software Development Life Cycle (SDLC) Process* (articles). https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/sdlc.html (2008-2009).
- Graham, Dan. *Introduction to the CLASP Process*. https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/requirements/548-BSI.html (2006).
- Ingalsbe, Jeffrey A.; Kunimatsu, Louis; Baeten, Tim; & Mead, Nancy R. "Threat Modeling: Diving into the Deep End." *IEEE Software* 25, 1 (January/February 2008). https://buildsecurityin.us-cert.gov/bsi/resources/articles/932-BSI.html

  This document describes industry experience with threat modeling.
- Mansourov, Nicolai & Campara, Djenana. *System Assurance: Beyond Detecting Vulnerabilities.* Elsevier, 2011. http://www.elsevierdirect.com/ISBN/9780123814142/System-Assurance

  Abstract from Publisher

  In this day of frequent acquisitions and perpetual application integrations, systems are often an amalgamation of multiple programming languages and runtime platforms using new and legacy content. Systems of such mixed origins are increasingly vulnerable to defects and subversion. System Assurance: Beyond Detecting Vulnerabilities addresses these critical issues.
- McGraw, Gary; Chess, Brian; & Migues, Sammy. *Building Security In Maturity Model (BSIMM)*. http://www.bsimm.com/ (2010).

  The BSIMM describes best practices based on a survey of a large number of organizations.
- OpenSAMM Project. *Software Assurance Maturity Model (SAMM) v1.0.* http://www.owasp.org/index.php/Category:Software_Assurance_Maturity_Model (2009).

  This document provides a maturity model that is specific to software assurance.
- CERT. *SQUARE* (educational materials for download). Software Engineering Institute, Carnegie Mellon University. http://www.cert.org/sse/square.html (2010).

This document includes a set of lectures and notes with an overview of security requirements engineering and details of the SQUARE Method. A team project for SQUARE is also included.

- Howard, Michael & Lipner, Steve. The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software. Microsoft Press, 2006. An online version of the Microsoft SDL is available at http://www.microsoft.com/security/sdl/.
- Massacci, Fabio; Mylopoulos, John; & Zannone, Nicola. "Computer-Aided Support for Secure Tropos." Automated Software Engineering 14, 3 (September 2007): 341–364.
- Zannone, Nicola. "The Si* Modeling Framework: Metamodel and Applications." International Journal of Software Engineering and Knowledge Engineering 19, 5 (August 2009): 727–746.

5.6 Assignments

Students can work on a team project for early assurance life-cycle activities, particularly those that result in a consistent and complete set of assurance requirements. Such a project is provided in the SQUARE educational workshop materials. To the extent possible, the assignments can be related to the project.

Additional assignments that do not exactly fit the project can be done as standalone ones. Examples of project and individual assignments are given in the table below. Assignments are discussed and assigned in the week shown and due the following week.

5.7 In-Class Activities

In-class activities and demonstrations are described in the suggested schedule below. Note that depending on the time available, the number of activities could be increased or decreased.

5.8 Suggested Schedule

The syllabus in Table 4 below defines in-class discussions and other activities that are intended to reinforce lecture material and homework assignments.

Table 4: Syllabus for the Assured Software Development (ASD1) Course

Week	Topic	In-Class Activities	Suggested Readings	Assignment
1	**Software process overview** Life-cycle processes including spiral, waterfall, agile, and associated activities. Discuss the entire spectrum of life-cycle activities including evolution.	Discuss when a specific life-cycle process is particularly appropriate.	• [Merkow 2010] • [Howard 2006] Chapters 1-4 • [Allen 2008] Chapter 1 • [Ahern 2008] Chapter 1 • [Mansourov 2011] Chapters 1, 2, 3 • The instructor may choose not to assign all of these readings, depending on how soon the instructor wants to start emphasizing assurance vs. ordinary development.	• Trade off life-cycle processes to determine which one might be the best fit for a specific project. Consider processes such as waterfall, spiral, agile, iterative development, and so on. • Project: Assign roles for the project. Select a life-cycle process for the project. Discuss the pros and cons of each process approach.
2	Discuss supply chain, acquisition, and service. Discuss Common Criteria.	• Discuss supply chain risks and the pros and cons of rigorous processes such as Common Criteria. • Discuss situations when such a process is called for.	• [Ellison 2010b] • [Mead 2008] • [Mansourov 2011] Chapter 4	• Modify a standard life-cycle process, such as agile or spiral, to emphasize assurance. • Project: Identify uses of acquired/COTS software on the project and perform tradeoff analysis to select COTS. Indicate where/how Common Criteria might be applied.
3	Introduce processes that are specific to software assurance, such as CLASP and Secure Tropos.	Discuss the pros and cons of standard development process models when applied to assured systems.	• [Mouratidis 2010] • [Haley 2008] • [Mouratidis 2007] Pages 16-43 • [Graham 2006]	Project: Sketch out how CLASP or Secure Tropos could be applied to the project.
4	Teach BSIMM, SAFECode and OWASP best practices.	Discuss the pros and cons of security process models and security maturity models, such as CLASP, BSIMM, and SAMM, and best practices such as those described by SAFECode.	• [McGraw 2010] • [OpenSAMM Project 2009] • [SAFECode 2008a] • [OpenSAMM 2009] • [SAFECode 2008b] • Students should look at the top level of each of these references and compare the elements. They do not need to read hundreds of pages of detailed discussion.	Project: Determine which of the BSIMM best practices should be applied to the project.

Week	Topic	In-Class Activities	Suggested Readings	Assignment
5	Methods for evaluation of environments, languages, and tools.		• [Howard 2006] Chapter 21 • [Massacci 2007] • [Zannone 2009]	Project: Select an environment that could be used on the project. Identify tools that could be used.
6	Teach quality factors and quality assessment methods as they relate to early life-cycle activities. Identify the different types of stakeholders and also likely developer roles.	Role-play part of a QAW with some students playing developer roles and others playing stakeholder roles.	[Barbacci 2003]	Project: Perform a QAW to understand the importance of security relative to other quality factors.
7	Teach practices that improve assurance at each life-cycle phase. Include requirements engineering, architecture, and design. Include coding, test, evolution, acquisition, and retirement. Teach practices such as threat modeling, assurance risk assessment, attack trees, and misuse and abuse cases. (carries into the following week).	Discuss ways in which the Microsoft SDL could be applied in the early life-cycle phases.	• [Allen 2008] Chapters 2 & 3 • [Ahern 2008] Chapter 2 • [Howard 2006] Chapters 8-9 • [Mansourov 2011] Chapter 5 • [Merkow 2010] • [DHS 2008-2009b] SDLC https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/sdlc.html Risk assessment https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/risk/250-BSI.html • Instructors may choose not to assign all of these readings, depending on which practices are viewed as most important or most practical.	Project: Perform risk analysis specifically for security.
8	Teach practices such as threat modeling, assurance risk assessment, attack trees, and misuse/ abuse cases.	Discuss security risk analysis results.	• [Allen 2008] Chapter 3 • [DHS 2010] • [DHS 2008-2009a] Attack Patterns https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/attack.html • [Ingalsbe 2008] • [Alexander 2003]	• Develop misuse cases for a small problem. • Project: Perform threat modeling.

Week	Topic	In-Class Activities	Suggested Readings	Assignment
9	Tools that can be used in the early life-cycle phases, either as part of a larger environment such as Rational or standalone tools such as SQUARE		• [IBM 2011] Rational [DOORS, RequisitePro] • [CERT 2010c] • [DHS 2010] • The instructor may choose to add articles/material on tools and environments.	Project: Develop misuse and abuse cases.
10	Teach a variety of elicitation methods, including those that are generic and those that are specific to security requirements.	Perform security requirements elicitation activity, with some students playing the roles of requirements engineers, and others playing the role of stakeholders.	• [Allen 2008] Chapter 3 • [DHS 2008-2009a] Requirements elicitation https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/requirements/533-BSI.html https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/requirements/532-BSI.html	Project: Apply requirements elicitation method(s) to identify requirements.
11	Ways of classifying or categorizing security requirements. How to distinguish requirements from architectural and design features, and mechanisms		• [Allen 2008] Chapter 3 • [CERT 2010c] Report material on categorization • If this does not take the whole week, the Week 12 material, which can easily take more than one week, can be started early.	Project: Perform requirements feasibility analysis and classify or categorize requirements to ensure they are valid.
12	Requirements prioritization methods, including group methods, formal cost/benefit tradeoff analysis, and factoring risk into the tradeoff analysis process		• [Allen 2008] Chapter 3 • [DHS 2008-2009a] Articles on requirements prioritization https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/requirements/545-BSI.html https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/requirements/534-BSI.html https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/requirements/1155-BSI.html https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/requirements/552-BSI.html	Project: Prioritize assurance requirements using cost/value or risk methods.

Week	Topic	In-Class Activities	Suggested Readings	Assignment
13	Requirements peer reviews, inspections, and traceability of requirements to assets and security goals	Perform part of a security requirements inspection, emphasizing traceability and missing requirements.	• [Wikipedia 2011] • [CERT 2010c] Assets and security goals http://www.sei.cmu.edu/library/abstracts/reports/05tr009.cfm http://www.sei.cmu.edu/library/abstracts/reports/05tr009.cfm	Project: Conduct a requirements inspection that is specific to assurance on the student team projects. Teams participate in inspection of other teams' requirements. Perform traceability.
14	**Project presentations and exam**			

6 Assured Software Development 2 (ASD2) Course

6.1 Catalog Description

This course covers rigorous methods for specifying assurance requirements and for architecting and designing software and systems to meet those requirements. Such methods include requirements specification; applying security principles; threat identification, characterization, and modeling; misuse/abuse cases; architectural risk analysis; architectural vulnerability assessment; and technology-specific security guidelines.

6.2 Prerequisites/Corequisites

The following can be either a prerequisite or a corequisite:

- ASD1 course

6.3 Expected Student Outcomes

After completing this course, students will be able to

1. specify and validate requirements for assured software
2. develop architectures that demonstrate that software and systems will satisfy their assurance requirements
3. design software and systems that fulfill architectural specifications for assurance
4. evaluate the capabilities and limitations of technical environments, languages, and tools when developing assured software
5. use assurance architecture and design methods, such as architectural risk analysis (including attack resistance, attack tolerance, and attack resilience), threat modeling, attack patterns, misuse/abuse cases, attack surface, design principles (such as least privilege and failing securely), and technology-specific guidelines
6. apply security technologies in specifying requirements and in developing architectures and designs, such as encryption, fault tolerance, intrusion detection, access controls, and authentication
7. understand design approaches and tactics for achieving quality attributes, including security

6.4 List of Topics

Topics on software assurance integration into SDLC phases (Appendix A, Section 1.2.2) include

- how to integrate assurance practices into typical life-cycle phases (requirements engineering, architecture, and design)

Topics on the evaluation of assurance technology (Appendix A, Section 6.1.1) include

- how to evaluate capabilities and limitations of technical environments, languages, and tools with respect to creating assured software functionality and security. Of particular interest are environments that support assurance, as well as languages that provide fewer opportunities to insert vulnerabilities, and tools used to improve assurance at various phases in the life cycle. In this course, we will be particularly interested in methods, tools, and environments that support specification, architecture, and high-level design.

Topics on the improvement of assurance technology (Appendix A, Section 6.1.2) include

- how to assess and recommend improvements in assurance technology as needed within project constraints, including cost, schedule, functionality, and quality factors. In this course, we will be particularly interested in improvements in technologies that support specification, architecture, and high-level design.

Topics on assured software development methods (Appendix A, Section 6.2.1: specification, architecture, and design) include

- how to inspect or otherwise verify specifications, architectures, and designs. Also, rigorous methods for developing assured system and software specifications, architectures, and high-level designs and how to apply those methods. This includes the use of formal specification languages that are specific to assurance, the use of architectural models, and the ability to perform architectural risk analysis and architectural tradeoff analysis. It also includes the use of design models and rigorous design languages to document and validate design. Students should understand how to do traceability from requirements through design and be able to extend this knowledge to code.

6.5 Sources

6.5.1 Primary

- Allen, Julia H.; Barnum, Sean; Ellison, Robert J.; McGraw, Gary; & Mead, Nancy R. Ch. 4, "Secure Software Architecture and Design," 115-150. *Software Security Engineering: A Guide for Project Managers*. Addison-Wesley Professional, 2008.

  Abstract from publisher

  Software that is developed from the beginning with security in mind will resist, tolerate, and recover from attacks more effectively than would otherwise be possible. While there may be no silver bullet for security, there are practices that project managers will find beneficial. With this management guide, you can select from a number of sound practices likely to increase the security and dependability of your software, both during its development and subsequently in its operation.

  Software Security Engineering draws extensively on the systematic approach developed for the Build Security In (BSI) Web site. Sponsored by the Department of Homeland Security Software Assurance Program, the BSI site offers a host of tools, guidelines, rules, principles, and other resources to help project managers address security issues in every phase of the

software development life cycle (SDLC). The book's expert authors, themselves frequent contributors to the BSI site, represent two well-known resources in the security world: the CERT Program at the Software Engineering Institute (SEI) and Cigital, Inc., a consulting firm specializing in software security. This book will help you understand why

  - Software security is about more than just eliminating vulnerabilities and conducting penetration tests
  - Network security mechanisms and IT infrastructure security services do not sufficiently protect application software from security risks
  - Software security initiatives should follow a risk-management approach to identify priorities and to define what is "good enough"—understanding that software security risks will change throughout the SDLC
  - Project managers and software engineers need to learn to think like an attacker in order to address the range of functions that software should not do, and how software can better resist, tolerate, and recover when under attack

- McGraw, Gary; Chess, Brian; & Migues, Sammy. *Building Security In Maturity Model (BSIMM)*. http://www.bsimm.com/ (2010).

  Software Security Framework describes twelve practices organized into four domains. These practices are used to organize the 109 BSIMM activities. All examples are real examples drawn from field observation.

  The software security best practices (or "touchpoints") have their basis in good software engineering and involve explicitly pondering security throughout the software development lifecycle. This means knowing and understanding common risks (including implementation bugs and architectural flaws), designing for security, and subjecting all software artifacts to thorough, objective risk analyses and testing. The practices include code review using static analysis tools, architectural risk analysis, penetration and risk-based security testing, abuse case development, security requirements and operations.

6.5.2 Secondary

Note: The list below is extensive, but not exhaustive, including several sources with similar material. It is up to the course instructor to select the source that best fits the specific delivery strategy.

- Merkow, Mark S. & Raghavan, Lakshmikanth. *Secure and Resilient Software Development.* CRC Press, 2010.
- Mouratidis, Haralambos & Paolo, Giorgini. Integrating Security and Software Engineering: Advances and Future Visions. Idea Group Publishing, 2007.

  This book draws upon research and techniques from a range of software engineering activities including requirements engineering and specification, software patterns and design, and methods and process of model-driven development.

- Pressman, Roger S., Software Engineering: A Practitioner's Approach, 6th ed. McGraw Hill, 2009.

- Howard, Michael & Lipner, Steve. *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software*. Microsoft Press, 2006. An online version of the Microsoft SDL is available at http://www.microsoft.com/security/sdl/.
- Altran Group. *Correctness by Construction*. http://www.altran-praxis.com/cbyc.aspx (Accessed 2010).
- SWIFT System. *Swift: making web applications secure by construction*. http://www.cs.cornell.edu/jif/swift/ (Accessed 2010).
- Schumacher, Markus; Fernandez-Buglioni, Eduardo; Hybertson, Duane; Buschmann, Frank; & Sommerlad, Peter. *Security Patterns: Integrating Security and Systems Engineering*, Wiley Series in Software Design Patterns, 2006.
- Department of Homeland Security (DHS). *Build Security In, Best Practices*. https://buildsecurityin.us-cert.gov/bsi/articles/best-practices.html (2008–2009).
- Berg, Clifford J. High Assurance Design: Architecting Secure and Reliable Enterprise Applications. Addison-Wesley Professional, 2005.
- Kazman, Rick; Klein, Mark H.; & Clements, Paul C. *ATAM: Method for Architecture Evaluation* (CMU/SEI-2000-TR-004). Software Engineering Institute, Carnegie Mellon University, 2000. http://www.sei.cmu.edu/library/abstracts/reports/00tr004.cfm
- The MITRE Corporation (MITRE). *CAPEC: Common Attack Pattern Enumeration and Classification*. http://capec.mitre.org/ (2010).
- National Institute of Standards and Technology (NIST). *SAMATE: Software Assurance Metrics and Tool Evaluation*. http://samate.nist.gov/Main_Page.html (2005).
- CERT. *Insider Threat and the Software Development Life Cycle* (podcast). Software Engineering Institute, Carnegie Mellon University. http://www.cert.org/podcast/show/20080304cappelli.html (2008).
- CERT. *Integrating Privacy Practices into the Software Development Life Cycle* (podcast).Software Engineering Institute, Carnegie Mellon University. http://www.cert.org/podcast/show/20091222hood.html (2009).
- CoFI. *CASL from CoFI*. http://www.informatik.uni-bremen.de/cofi/wiki/index.php/CASL (2008).
- Fitzgerald, John. *Welcome to the VDM Portal*. http://www.vdmportal.org/twiki/bin/view (2010).
- Community Z Tools. *Overview*. http://czt.sourceforge.net/ (2007).
- The ZETA System. *Overview*. http://uebb.cs.tu-berlin.de/zeta/ (Accessed 2010).
- Jacky, Jonathan. *The Way of Z: Practical Programming with Formal Methods*. Cambridge University Press, 1996.
- Blanchette, Stephen, Jr.; Crosson, Steven; & Boehm, Barry. *Evaluating the Software Design of a Complex System of Systems* (CMU/SEI-2009-TR-023, ESC-TR-2009-023). Software En-

gineering Institute, Carnegie Mellon University, 2009. http://www.sei.cmu.edu/library/abstracts/reports/09tr023.cfm

- Mellado, Daniel; Fernández-Medina, Eduardo; & Piattini, Mario. "A Comparison of Software Design Security Metrics," 236–242. *Proceedings of the Fourth European Conference on Software Architecture*. Copenhagen, Denmark, Aug. 2010. ACM, 2010.
- Leroy, Xavier. "Computer Security from a Programming Language and Static Analysis Perspective," 1–9. *Proceedings of the 12th European Conference on Programming*. Warsaw, Poland, April 2003. Springer-Verlag, 2003.
- Myers, Andrew. *PLD'06 Tutorial T1: Enforcing and Expressing Security with Programming Languages*. http://www.cs.cornell.edu/andru/pldi06-tutorial/06jun-pldi-tutorial.pdf (2006).
- Bagheri, Hamid & Mirian-Hosseinabadi, Seyed-Hassan. "Injecting Security as Aspectable NFR into Software Architecture," *Proceedings of the 14th Asia-Pacific Software Engineering Conference*. Nagoya, Japan. Dec. 2007. IEEE Computer Society Press, 2008.
- Ellison, Robert J.; Goodenough, John B.; Weinstock, Charles B.; & Woody, Carol. *Evaluating and Mitigating Software Supply Chain Security Risks* (CMU/SEI-2010-TN-016). Software Engineering Institute, Carnegie Mellon University, 2010. http://www.sei.cmu.edu/library/abstracts/reports/10tn016.cfm
- Bode, Stephan; Fischer, Anja; Kühnhauser, Winfried; & Riebisch, Matthias. "Software Architectural Design Meets Security Engineering," *Proceedings of the 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*. San Francisco, CA, April 2009. IEEE Computer Society Press, 2009.
- Ray, Arnab & Cleaveland, Rance. "A Software Architectural Approach to Security By Design," *Proceedings of the 30th International Computer Software and Applications Conference*. Chicago, Ill, Sept. 2006. IEEE Computer Society Press, 2006.
- Hansson, Jörgen; Wrage, Lutz; Feiller, Peter H.; & Morley, John. "Architectural Modeling to Verify Security and Nonfunctional Behavior." *IEEE Security & Practice 8*, 1: (Jan./Feb. 2010): 43–49.
- Rehman, S. & Mustafa, K. "Research on Software Design Level Security Vulnerabilities." *ACM SIGSOFT Software Engineering Notes 34*, 6 (November 2009): 1–5.
- The Open Group. *TOGAF*. http://www.opengroup.org/togaf/ (2010).

  TOGAF is an industry standard architecture framework that may be used freely by any organization wishing to develop information systems architecture for use within that organization. TOGAF has been developed and continuously evolved since the mid-90s by representatives of some of the world's leading IT customer and vendor organizations, working in The Open Group's Architecture Forum. Details of the Forum and its plans for evolving TOGAF in the current year are provided on the Architecture Forum website. http://www.opengroup.org/architecture/.

  Note: If your school can afford membership ($1k or $2.5k, depending on the level), you can find interesting materials in the TOGAF Security Forum.

6.6 Assignments

Assignments are discussed and assigned in the week shown and due the following week.

6.7 In-Class Activities

In-class activities and demonstrations are described in the suggested schedule below. A reading assigned in advance should facilitate instruction and in-class discussions. One of the possible solutions for out-of-classroom work is to assign a semester-long, small-group project that will carry through the early part of the development life cycle (specification, architecture, design), while focusing on security and assured development. The biweekly assignments could constitute partial deliverables to be collected and edited into the final project. Note that depending on the time available, the number of activities could be increased or decreased.

Other proposed team or individual activities may include

- Evaluate capabilities and limitations of a selected development environment (language, tool).
- Assess and recommend improvements in assurance technology.
- Given an example of software documentation (SRS, SDD), create a traceability matrix.
- Given an example of a project, discuss the applicable constraints considering cost, schedule, functionality, and quality.
- Given an example of a software specification, carry out a formal requirements inspection.
- Given an example of software design, carry out a formal design inspection.
- For a given software system architecture, perform architectural risk analysis from a security perspective.
- Specify dependability requirements for a hypothetical system based on background information from the AA course.

6.8 Suggested Schedule

The syllabus in Table 5 below defines in-class discussions and other activities that are intended to reinforce lecture material and homework assignments.

The students are assumed to have access to a software development environment and tools that allow them to create software projects. Examples of such tools are

- MagicDraw, No Magic Inc. *http://www.magicdraw.com/*
- *IBM Rational Modeler http://www.ibm.com/developerworks/downloads/r/modeler/*
- Together Architect, Borland *http://www.borland.com/us/products/together/*
- Enterprise Architect, Sparx Systems *http://www.sparxsystems.com/products/ea/*
- *Artisan Studio, Artisan Studio Uno, http://www.artisansoftwaretools.com/studiouno*
- IBM Rational Rhapsody Developer, *http://www.ibm.com/developerworks/downloads/r/rhapsodydeveloper/*

- IBM Rational Software Architect V8
  http://www.ibm.com/developerworks/downloads/r/architect/

Table 5: Syllabus for the Assured Software Development (ASD2) Course

Week	Topic	In-Class Activities	Suggested Reading	Assignment
1	Concepts of assured development life cycle	• Discussion of course objectives, content, and activities and brief review of basics of software engineering from the assurance viewpoint (definitions, practices, process, standards) • Class introductions and assessment of the audience	• [Bishop 2002] Chapter 18 • [Allen 2008] Chapters 1, 2 • [Pressman 2009] Chapter 1 • [Merkow 2010] Chapter 3 • [Mouratidis 2007] Chapter 1 • [CERT 2008]	
2	Assurance issues in front-end development life cycle (specification, architecture, design)	• Discussion of components of computer security, threats, policies and mechanisms, the role of trust, assurance, operational issues, and human issues • Discussion of threat identification and the protection models • Form project teams and provide guidance for team project.	• [Bishop 2002] Chapter 19 • [Allen 2008] Chapters 3,4 • [Pressman 2009] Chapters 3,4 • [Merkow 2010] Chapter 5 • [DHS 2008-2009a] • [Mellado 2010] • [CERT 2009]	
3	Software development environments supporting specification, architecture, and high-level design	• Discussion of software development environments • Discussion of security, confidentiality and integrity policies • Class exercise: Create small specification based on user need document.	• [Pressman 2009] Chapters 7, 8 • [Merkow 2010] Chapter 5 • [McGraw 2010] • [Bagheri 2008] • [Mouratidis 2007] Chapters 1, 2	Project Part 1: Create software specification of an example application.
4	Tools support for assured software development	• Discussion of tools supporting assured development and their applicability to address misuse/abuse cases • Demonstration of selected tool and development environment	• [Pressman 2009] Chapter 9 • selected readings from SAMATE and CAPEC	Project Part 2: Use example specification to develop simple throw-away design prototype in available environment.

Week	Topic	In-Class Activities	Suggested Reading	Assignment
5	Languages review	• Discussion of properties of modern languages from security perspective • Identification of potential security flaws in Java, C/C++, Ada, PHP, etc. • Class exercise: Experiment with a selected development environment.	• [Leroy 2003] • [Myers 2006]	
6	Project constraints aspects: cost, schedule, functionality, and quality factors	• Discussion of project constraints from the security perspective • Class exercise: Identify simple application constraints.	• [Allen 2008] Chapter 7 • [McGraw 2010]	Project Part 3: Analyze application constraints.
7	Formal specification languages and technologies	• Discussion of formal specification and their impact on assured development • Class exercise: Create small formal specification.	• [Bishop 2002] Chapter 20 • [Pressman 2009] Chapter 28 • selected readings on formal tool(s) [ZETA System 2010] [Community 2007] [CoFI 2008] [Fitzgerald 2010] [Jacky 1996]	
8	Improvements in technologies to support specification, architecture, and high-level design	• Discussion of available technologies • Class exercise: Validate the example application design and complete security report.	• [Schumacher 2006] Chapters 8-10 • [Mouratidis 2007] Chapter 12 • [Altran 2010] • [SWIFT System 2010] • [McGraw 2010]	Project Part 4: Use formal specification to validate requirements of the example application.
9	Architectural models and viewpoints	• Discussion of the software architecture views with a focus on security and architectural vulnerability • Discussion of product lines	• [Pressman 2009] Chapter 10 • [Schumacher 2006] Chapters 1, 3-5 • [Ellison 2010a] • [Mouratidis 2007] Chapters 5-8	

Week	Topic	In-Class Activities	Suggested Reading	Assignment
10	Architectural risk and tradeoff analysis	• Discussion of architectural risk analysis (attack resistance, tolerance, and resilience) and threat modeling • Class exercise: Perform mock-up risk analysis of multiple threats.	• [Allen 2008] Chapter 6 • [Mouratidis 2007] Chapter 9 • [Bode 2009] • [Howard2006] Chapters 8, 9 • [Merkow 2010] Chapter 11 • [Kazman 2000] • [McGraw 2010]	Project Part 5: Use a tool to model selected threats and evaluate choices of architectural models for the application.
11	Methods and technologies for developing assured system and software specifications, architectures, and high-level designs	• Discussion of design principles (least privilege, secure failing) and technology-specific guidelines • Class exercise: Use tool to model a simple application.	• [Ray 2006] • [Hansson 2010] • [Howard 2006] Part II • [Schumacher 2006] Chapters 12, 13 • [Merkow 2010] Chapter 7 • [McGraw 2010]	
12	Design models and languages	• Discussion of modeling languages with a focus on their features supporting assured development	• [Rehman 2009]	Project Part 6: Use tool to complete design of the example application.
13	Design validation and software inspections	• Discussion of specification and design inspections • Class exercise: Carry out a mock-up design inspection.	• [Pressman 2009] Chapters 13, 14 • [Blanchette 2009] • [Schumacher 2006] Chapter 11 • [McGraw 2010]	
14	**Final project report and presentation**			

7 Assured Software Development 3 (ASD3) Course

7.1 Catalog Description

This course covers rigorous methods, techniques, and tools for developing secure code. Such methods include code analysis for commonly known vulnerabilities, source code review using static analysis tools, and known, language-specific practices for producing secure code.

This course also covers rigorous methods and tools for inspecting, testing, verifying, and validating software and systems to demonstrate that they meet functional and security requirements. Students will learn methods for verification and validation for security assurance and how security vulnerabilities can differ from programming errors. Team inspections and correctness verification methods will be covered. Testing techniques will include threat- and attack-based testing, functional testing, risk- and usage-based testing, stress testing, black- and white-box testing, and penetration testing.

7.2 Prerequisites/Corequisites

- Some programming experience, preferably in C or C++, is needed as a prerequisite.
- The ASD1 course can be either a prerequisite or a corequisite. In either case, Weeks 3 and 4 may be replaced by a more extended treatment of other parts of the course.

7.3 Expected Student Outcomes

After completing this course, students will be able to

1. develop software that does not contain known vulnerabilities such as incorrect or incomplete input validation, poor or missing exception handling, buffer overflows, SQL injection, and race conditions
2. use methods, techniques, and tools that demonstrate that developed software meets its functionality and security requirements and implements its security architecture and design specifications
3. understand how to apply team inspections to validate the functionality and security properties of software
4. understand methods for correctness verification of critical software components
5. understand how testing for security differs from traditional testing
6. test software to ensure that assurance requirements are met using a variety of methods, techniques, and tools
7. use threat models, attack patterns, and misuse/abuse cases during software and system testing
8. maintain software to continue to meet its functionality and security requirements

7.4 List of Topics

Topics on rigorous methods for system implementation, verification, and testing to develop assured software (Appendix A, Section 6.2) include

- motivation: common vulnerabilities and weaknesses of code
- common pitfalls of string processing: termination of string values, memory management, and stack smashing
- data representation and pointer issues: unsafe conversions, pointer arithmetic, and code optimization
- vulnerabilities in modern languages, such as Java, C#, and PHP
- recommended coding practices
- code inspections and other code-reading techniques
- pre- and post-conditions for specifying the behavior of code units and proofs of correctness to show that code meets its specified requirements
- unit, integration, system, and regression testing, including assurance needs and techniques
- other static analysis methods and tools: detecting memory leaks, and buffer overflow vulnerabilities
- other dynamic analysis methods and tools: fuzzers, penetration testing, and other attack-based methods

Topics on assurance aspects of software maintenance and evolution (Appendix A, Section 6.2) include

- methods for restructuring code to improve understandability
- regression testing
- use of configuration management systems and processes

7.5 Sources

7.5.1 Primary

- Merkow, Mark S. & Raghavan, Lakshmikanth. *Secure and Resilient Software Development.* CRC Press, 2010.

  Abstract from publisher

  Although many software books highlight open problems in secure software development, few provide easily actionable, ground-level solutions. Breaking the mold, Secure and Resilient Software Development teaches you how to apply best practices and standards for consistent and secure software development. It details specific quality software development strategies and practices that stress resilience requirements with precise, actionable, and ground-level inputs.

Providing comprehensive coverage, the book illustrates all phases of the secure software development life cycle. It shows developers how to master non-functional requirements, including reliability, security, and resilience. The authors provide expert-level guidance through all phases of the process and supply many best practices, principles, testing practices, and design methodologies.

- Seacord, Robert C. *Secure Coding in C and C++*. Addison-Wesley, 2005. http://www.sei.cmu.edu/library/abstracts/books/0321335724.cfm

  Abstract from publisher

  Commonly exploited software vulnerabilities are usually caused by avoidable software defects. Having analyzed nearly 18,000 vulnerability reports over the past ten years, the CERT/Coordination Center (CERT/CC) has determined that a relatively small number of root causes account for most of them. This book identifies and explains these causes and shows the steps that can be taken to prevent exploitation. Moreover, this book encourages programmers to adopt security best practices and develop a s security mindset that can help protect software from tomorrow's attacks, not just today's.

  Drawing on the CERT/CC's reports and conclusions, Robert Seacord systematically identifies the program errors most likely to lead to security breaches, shows how they can be exploited, reviews the potential consequences, and present secure alternatives.

  Coverage includes technical detail on how to

  - Improve the overall security of any C/C++ application
  - Thwart buffer overflows and stack-smashing attacks that exploit insecure string manipulation logic
  - Avoid vulnerabilities and security flaws resulting from the incorrect use of dynamic memory management functions
  - Eliminate integer-related problems: integer overflows, sing errors, and truncation errors
  - Correctly use formatted output functions without introducing format-string vulnerabilities
  - Avoid I/O vulnerabilities, including race conditions

  Secure Coding in C and C++ presents hundreds of examples of secure code, insecure code, and exploits, implemented for Windows and Linux. If you're responsible for creating secure C or C++ software—or for keeping it safe—no other book offers you this much detailed, expert assistance.

7.5.2 Secondary

- The MITRE Corporation (MITRE). *Common Weakness Enumeration*. http://cwe.mitre.org/ (2010).
- Grembi, Jason. "Secure Software Development - A Security Programmer's Guide." Tutorial at *11th Semi-Annual Software Assurance Forum*. Arlington, VA, November 2009. Software Engineering Institute, Carnegie Mellon University, 2009. https://www.vte.cert.org/vteweb/go/2699.aspx

- Gerhart, Susan; Hogle, Jan; & Crandall, Jedidiah. *How Do Buffer Overflow Attacks Work?* http://nsfsecurity.pr.erau.edu/bom/ (2002).

  This document provides a nice introduction to buffer overflows and stack-smashing using animation and student exercises.

- CERT. *CERT Secure Coding Standards*. https://www.securecoding.cert.org/ (2010).

  This document includes advice for Java, as well as C and C++.

- Miller, Barton P.; Cooksey, Gregory; & Moore, Fredrick. "An Empirical Study of the Robustness of MacOS Applications Using Random Testing." *ACM SIGOPS Operating Systems Review 41*, 1 (January 2007): 78-86.

  This document describes the results of fuzz testing many common applications, including a review of earlier testing of Unix and Windows systems.

- Golze, Andreas; Sarbiewski, Mark; & Zahm, Alain. Optimize Quality for Business Outcomes: A Practical Approach to Software Testing. Wiley Publishing, 2008.

  Chapter 8 on Security Testing is especially useful.

- Howard, Michael & LeBlanc, David. *Writing Secure Code*, 2nd ed. Microsoft Press, 2003.
- Howard, Michael; LeBlanc, David; & Viega, John. *19 Deadly Sins of Software Security*. McGraw-Hill, 2005.

7.6 Assignments

Students should complete individual assignments as described in the suggested schedule below. They should also work on a team project that includes implementing a simple system, inspecting it, performing static analysis, and finally testing the system. Assignments are discussed and assigned in the week shown and due the following week.

7.7 In-Class Activities

The buffer overflow demos and exercises provided at http://nsfsecurity.pr.erau.edu/bom are particularly helpful. In addition, it is very useful to conduct a demonstration code inspection with the whole class acting as reviewers. This gives the instructor the opportunity to demonstrate and describe expected behavior before and during the inspection. Many of the homework assignments may also be conducted as in-class activities. All in-class activities and demonstrations are described in the suggested schedule below. Note that depending on the time available, the number of activities could be increased or decreased.

7.8 Suggested Schedule

The syllabus in Table 6 below defines in-class discussions and other activities that are intended to reinforce lecture material and homework assignments.

Table 6: Syllabus for the Assured Software Development (ASD3) Course

Week	Topic	In-Class Activities	Suggested Readings	Assignment
1	**Introduction:** • Overview of vulnerabilities and their costs • Properties of secure and resilient software		[Merkow 2010] Chapters 1, 2	• Write a simple program to be checked for vulnerabilities later. • Project: Elicit simple requirements.
2	**Vulnerabilities:** • CWE/SANS Top 25 Most Dangerous Programming Errors • Security Concepts		• [Merkow 2010] Appendix A • [Seacord 2005] Chapter 1	• Read "Seven Pernicious Kingdoms" paper and categorize several attacks according to this taxonomy. • Project: Review requirements.
3	**General Strategies:** • Security and resilience throughout the life cycle • Attack surfaces and security perimeters • OWASP Best Practices		[Merkow 2010] Chapters 3, 4	• Categorize risks by methods that can detect them and/or prevent them. • Project: Design solution.
4	**Development Practices:** • Best practices for Requirements, Architecture and Design (e.g., abuse/misuse cases, threat modeling, risk analysis, design reviews, defense in depth)		• [Merkow 2010] Chapter 5 • [Seacord 2005] Chapter 8	• Create abuse/misuse cases from requirements. • Describe risk mitigation strategies for threat models. • Describe recommended software development practices to create and maintain trustworthy code. • Project: Review design.
5	**Programming Practices:** • OWASP Top 10 Security Risks • OWASP Enterprise Security API • Cross-site scripting • Injection attacks • Authentication and session management		[Merkow 2010] Chapter 6	• Describe prevention methods for OWASP top 10. • Repair flawed code: authentication. • Project: Update design.

Week	Topic	In-Class Activities	Suggested Readings	Assignment
6	**Memory Management in C and C++:** • Common memory management errors (buffer overflow, stack smashing) • Input validation	View demos and perform exercises from http://nsfsecurity.pr.erau.edu/bom/.	[Seacord 2005] Chapters 2, 4	Review and repair flawed code for buffer overflow, input validation and memory management flaws.
7	**Strings, Pointers and Integers:** • Common string manipulation errors • Integer overflow vulnerabilities • Pointer subterfuge		[Seacord 2005] Chapters 3, 5	Review and repair flawed code for integer overflows and string processing insecurities.
8	**Other vulnerabilities in C and C++:** • Formatted I/O operations • File I/O race conditions (e.g., Time Of Use Time Of Check) • Other file system exploits		[Seacord 2005] Chapters 6, 7	• Review and repair flawed code for race condition insecurities (file I/O). • Project: Implementation due
9	**Inspections, proofs and code reading:** • Code-reading techniques • Formal code inspections • Program verification	Conduct demonstration code inspection.		• Review flawed code and identify vulnerabilities. • Carry out proof of correctness of a small program. • Review a flawed modification of a program. • Project: Code inspection
10	**Static Analysis:** • Types of static analysis • Modern analysis tools (e.g., Coverity, Fortify)		[Merkow 2010] Chapter 8	• Describe vulnerabilities that can be detected by static analysis (SA) and those that cannot. • For those vulnerabilities that cannot be detected by SA, describe mitigation strategies. • Project: Perform static analysis on code using a tool such as Fortify or Veracode.

Week	Topic	In-Class Activities	Suggested Readings	Assignment
11	**Testing:** • Best practices for unit testing • Penetration testing • Fuzzing • Overview of Common Criteria		[Merkow 2010] Chapter 9	• Describe penetration tests for a given system. • Create tests to demonstrate compliance with assurance requirements. • Perform fuzz testing of an example program. • Project: Review static analysis report.
12	**Insecurities in Java and other languages:** • Runtime environment • Coding practices • Overview of known vulnerabilities		[CERT 2011]	• Review and repair flawed Java program. • Project: Create test plan.
13	**Trends and Resources:** • Comprehensive, Lightweight Application Security Process (CLASP) • Certificates and courses in security and software assurance		[Merkow 2010] Chapters 10, 12	• Case study: Recommend process improvements using CLASP. • Create a website with links to resources useful in educating colleagues and in staying abreast of developments in software assurance. • Project: Conduct testing.
14	**Final Exam**			

8 Assurance Assessment (AA) Course

8.1 Catalog Description

This course covers the fundamentals of establishing a required level of software and system assurance and applying methods and determining measures to assess whether the required level of assurance has been achieved. Topics include assessment methods; defining product and process measures and other performance indicators; measurement processes and frameworks; performance indicators for business survivability and continuity; and comparing selected measures to determine whether the software/system meets its required level of assurance. These fundamentals are applied to newly developed software and systems, as well as during the acquisition of software and services.

8.2 Prerequisites

- the ability to develop a software module (design, code, test) using a contemporary programming language
- knowledge of the fundamentals of computer organization, operating systems, networks, and digital communications
- knowledge of general software engineering concepts and practices: the software life cycle, requirements engineering, software design, software construction, software verification and validation, and software development processes
- awareness of software security issues (e.g., properties, threats, and requirements)

8.3 Expected Student Outcomes

After completing this course, students will be able to

- specify a required level of assurance for a software product or system
- understand how to use a range of assessment methods including requirements validation, risk analysis, threat analysis, vulnerability assessment, and assurance cases
- define and develop key product and process measures and other performance indicators that can be used to validate a required level of assurance
- collect and report measures that indicate the extent to which software and systems have achieved their required level of assurance
- perform assurance assessment for newly developed software and systems
- perform assurance assessment for acquired systems and services, including developing service level agreements and monitoring performance against them

8.4 List of Topics

Topics on software assurance concepts, assurance assessment methods and processes, and measurement fundamentals (Appendix A, Sections 3.1, 3.2, 3.3) include

- introduction to software assurance
- software security fundamentals
- software assurance throughout the software development life cycle (SDLC)
- software assurance methods
- software measurement fundamentals
- product and process assurance measures
- software measurement processes and frameworks

Topics on software assurance assessment for existing software and for acquired software (Appendix A, Section 6.4) include

- software assurance assessment during system operation and maintenance
- software assurance assessment for acquired systems and software services

8.5 Sources

8.5.1 Primary

- Bishop, Matt. *Computer Security: Art and Science.* Addison-Wesley Professional, 2002.

  Abstract from publisher

  The importance of computer security has increased dramatically during the past few years. Bishop provides a monumental reference for the theory and practice of computer security. This is a textbook intended for use at the advanced undergraduate and introductory graduate levels, non-University training courses, as well as reference and self-study for security professionals. Comprehensive in scope, this covers applied and practical elements, theory, and the reasons for the design of applications and security techniques. Bishop treats the management and engineering issues of computer. Excellent examples of ideas and mechanisms show how disparate techniques and principles are combined (or not) in widely-used systems. Features a distillation of a vast number of conference papers, dissertations and books that have appeared over the years, providing a valuable synthesis. This book is acclaimed for its scope, clear and lucid writing, and its combination of formal and theoretical aspects with real systems, technologies, techniques, and policies.

- Kan, Stephen H. Metrics and Models in Software Quality Engineering, 2nd ed. Addison-Wesley, 2002.

  Abstract from publisher

  Our society has become increasingly reliant on software in the past decade; businesses have learned that measuring the effectiveness of software projects can impact the bottom line; and

quality is no longer an advantage in the software marketplace (it is a necessity). For these reasons, the demand for quality in software engineering has taken center stage in the twenty-first century. In this new edition, Stephen Kan presents a thoroughly updated overview and implementation guide for software engineers faced with the challenge of ensuring quality. The book balances theory, techniques, and real-life examples to provide practical guidelines in the practice of quality. Although there are equations and formulas presented, the book's focus remains on helping the reader understand and apply the metrics and models. With this book as a map, readers can navigate through the complex field of quality, and benefit their organization by improving their processes and products.

8.5.2 Secondary

- Alberts, Christopher; Allen, Julia; & Stoddard, Robert. *Integrated Measurement and Analysis Framework for Software Security* (CMU/SEI-2010-TN-025). Software Engineering Institute, Carnegie Mellon University, 2010.
- IEEE Standards Association (IEEE-SA). IEEE Std 1061–1998 IEEE Standard for a Software Quality Metrics Methodology. IEEE–SA, 1998.
- IEEE Standards Association (IEEE–SA) IEEE Std 1219-1998 *IEEE Standard for Software Maintenance*. IEEE–SA, 1998.
- IEEE Standards Association (IEEE–SA). IEEE Std 1062–1998 IEEE Recommended Practice for Software Acquisition. IEEE–SA, 1998.
- IEEE Standards Association (IEEE–SA). IEEE Std 15939–2007 IEEE Systems and Software Engineering—Measurement Process. IEEE–SA, 2007.
- Gold, Nicolas; Mohan, Andrew; Knight, Claire; & Munro, Malcolm. “Understanding Service-Oriented Software,” *IEEE Software 21*, 2 (March/April 2004): 71–77.

8.6 Assignments

Students are assigned reading each week in which new material is discussed. Assignments are discussed and assigned in the week shown and due the following week. For some weeks, individual software assurance assessment exercises are assigned, some based on in-class exercises. A major part of the course is a software assurance assessment team project.

- Teams will select an application that is under development and develop a measurement and analysis (MA) plan [Alberts 2010]. The teams will carry out the following activities: specify the objectives for measurement and analysis; specify the measures, analysis techniques, and mechanisms for data collection, data storage, data reporting, and feedback; determine how data will be collected, stored, analyzed, and reported; and decide how results can be used in making informed decisions and how to take appropriate corrective actions.
- The instructor will provide the teams with candidate applications. Bishop’s book [Bishop 2002] has some examples in Part 8.

8.7 In-Class Activities

In-class activities are detailed in the suggested schedule below. Most sessions consist of lecture and related discussion of various assurance assessment topics. Most sessions also include a group exercise. The exercise should take 15 to 30 minutes and be followed by a 15-to-20-minute class discussion. Note that depending on the time available, the number of activities could be increased or decreased.

8.8 Suggested Schedule

The syllabus in Table 7 below defines in-class discussions and other activities that are intended to reinforce lecture material and homework assignments.

Table 7: Syllabus for the Assurance Assessment (AA) Course

Week	Topic	In-Class Activities	Suggested Reading	Assignment
1	**Introduction to Software Assurance**	• Discussion of course objectives, content, and activities (including team project) • Discussion of software engineering: practices, processes, tools, and standards • Discussion of software assurance: definition, importance, current state • Group exercise based on one of the exercises in Chapter 17 of [Bishop 2003]	• [Bishop 2002] Chapter 18 • [Kan 2002] Chapters 1, 2	
2	**Software Security Fundamentals**	• Discussion of components of computer security, threats, policies and mechanisms, the role of trust, assurance, operational issues, and human issues • Discussion of access control matrix model and the protection model • Group exercise based on exercise 1 in Chapter 1 of [Bishop 2003]	[Bishop 2002] Chapters 1, 2, 3	Individual exercise based on exercise 2 in Chapter 1 of [Bishop 2003]

Week	Topic	In-Class Activities	Suggested Reading	Assignment
3	**Assurance through the SDLC**	• Discussion of the establishment and specification, throughout the SDLC, of the required or desired level of assurance for a specific software application, set of applications, or software-reliant system (and tolerance for same) • Discussion of security, confidentiality, and integrity polices • Group exercise based on exercise 1 or 2 in Chapter 3 of [Bishop 2003]	[Bishop 2002] Chapters 4, 5, 6	Complete and submit team project member information sheet.
4	**Assurance Methods 1**	• Discussion of assurance in requirements analysis and definition • Discussion of assurance in system and software design • Form project teams and provide guidance for team project proposal. • Group exercise based on exercise 5a in Chapter 19 of [Bishop 2003]	[Bishop 2002] Sections 19.1, 19.2	• Individual exercise based on exercise 5b in Chapter 19 of [Bishop 2003] • Team project work
5	**Assurance Methods 2**	• Discussion of assurance in implementation and integration • Discussion of assurance in operation and maintenance • Discussion of assurance of formal verification techniques • Group exercise based on a code review of a simple method [Bishop 2003]	[Bishop 2002] Sections 19.3, 19.4, 20.1, 20.2	• An exercise involving development of a set of test cases for the function specified in exercise 5a or 5b in Chapter 19 of [Bishop 2003] • Complete and submit team project proposal: team organization, problem need statement, project schedule and deliverables.

Week	Topic	In-Class Activities	Suggested Reading	Assignment
6	**Assurance Methods 3**	• Discussion of how various methods (such as validation of security requirements, risk analysis, threat analysis, vulnerability assessments and scans, and assurance evidence) can be used to assess the software assurance of a developing or existing system • Group exercise based on exercise 1 in Chapter 29 of [Bishop 2003]	[Bishop 2002] Chapter 29	• Exercise 2 in Chapter 29 of [Bishop 2003] • Team project work
7	**Measurement Fundamentals**	• Discussion of software quality, uses of measurement, basic measures, level of measurement, derived metrics, reliability and validity measures, measurement errors, correlation, causality, and software quality and measurement standards • Group exercise asking what practices should be part of the software process to ensure collection of one or more of the metrics listed in Table 1 in [Alberts 2010]	• [Kan 2002] Chapters 1, 3 • [Alberts 2010] Chapters 1, 2	Team project work
8	**Product and Process Assurance Measures**	• Discussion of the definition and development of key product and process measurements (and additional performance indicators) that can be used to validate the required level of software assurance appropriate to a given life-cycle phase • Discussion of the Goal-Question-Metric (GQM) techniques for determining metrics • Group exercise involving determining a software security metric using GQM	[Kan 2002] Chapters 4, 10, 15	Interim team project progress report

Week	Topic	In-Class Activities	Suggested Reading	Assignment
9	**Measurement Processes and Frameworks 1**	• Discussion of measurement processes and frameworks and their use in process/ practice assessment and in software assurance integration into the software development life cycle (SDLC) • Group exercise involving the development of a modest measurement process for the product development discussed in exercise 3 in Chapter 18 of [Bishop 2003]	• [Kan 2002] Chapters 5, 9 • [IEEE 1998a]	Team project work
10	**Measurement Processes and Frameworks 2**	• Discussion of establishing and sustaining a measurement program and process, planning for measurement, performing measurement, and evaluating the measurement process • Group exercise involving how a small to medium software producer should start up a metrics program. (This is to be done at a high conceptual level, simply indicating the types of activities that should be carried out.)	• [ISO 2007] • [Alberts 2010] Chapter 2 • [Kan 2002] Chapter 19	• On the basis of class discussion, complete the in-class exercise, providing detail about which measures would be made, and how they would be collected and used. • Team project work
11	**Assurance Assessment during System Operation and Maintenance**	• Discussion of assurance assessment methods and measures during the operational and maintenance phases of the SDLC • Group exercise involving how a small to medium company should assess the operation and maintenance of a software product produced in-house. (This is to be done at a high conceptual level, simply indicating the types of activities that should be carried out).	• [Kan 2002] Chapters 13, 14 • [IEEE 1998b] • [CMMI 2009] pp183-201	• On the basis of class discussion, complete the in-class exercise, providing detail about which measures would be made, and how they would be collected and used. • Team project work

Week	Topic	In-Class Activities	Suggested Reading	Assignment
12	**Assurance Assessment for Acquired Systems and Software Services**	• Discussion of assurance of software acquired through supply chains, vendors, and open sources, including developing requirements and assuring delivered functionality and security • Discussion of development of service level agreements for functionality and security with service providers and for monitoring compliance	• [Alberts 2010] Chapters 4, 5, 6 • [IEEE 1998c] • [Gold 2004] • [CMMI 2007] pp183-201	Team project work
13	**Project Presentations**	• Student teams make a 20-minute presentation about the results of their projects.		Final project report
14	**Final Exam**			

9 System Security Assurance (SSA) Course

9.1 Catalog Description

This course covers how to incorporate effective security technologies and methods into new and existing systems. Students will learn how to think like an attacker when planning a variety of attacks, including password cracking, escalation of privileges, denial of service, viruses, worms, Trojans, spyware, logic bombs, and other malicious code. They will learn the most effective methods for preventing or defeating these attacks and analyzing the threats that they pose. Students will understand their ethical responsibilities and obligations when developing, acquiring, and operating software and systems.

9.2 Prerequisites

- completion of the AA course
- basic programming skills in a commonly used, high-level language (e.g., C/C++, Java)

9.3 Expected Student Outcomes

After completing this course, students will be able to

1. describe the kinds of safety and security risks associated with critical infrastructure systems such as power, telecommunications, water, and air-traffic-control systems
2. understand the variety of methods attackers can use to damage software and its associated data via weaknesses in the system's design or code
3. analyze threats to software
4. describe and deploy appropriate countermeasures, such as layers, access controls, privileges, intrusion detection, encryption, and coding checklists
5. analyze threats to operational environments
6. design and plan for effective countermeasures such as access control, authentication, intrusion detection, encryption, and coding checklists
7. understand how physical security countermeasures, such as gates, locks, guards, and background checks, can address risks
8. understand how people who are knowledgeable about attack and prevention methods are obligated to use their abilities, for both legal and ethical reasons
9. understand the legal and ethical considerations involved in analyzing a variety of historical events and investigations

9.4 List of Topics

Topics on incorporating security technologies and methods into new and existing systems (Appendix A, Sections 5.1, 5.2, 5.3) include

- introduction to system security assurance
- attacker methods
- threat analysis
- security countermeasures
- planning for access control and authentication
- safety and security risks
- mitigating security risks
- legal and ethical issues

9.5 Sources

9.5.1 Primary

- Anderson, Ross J. Security Engineering: A Guide to Building Dependable Distributed Systems, 2nd ed. Wiley, 2008.

  Abstract from publisher

  The world has changed radically since the first edition was published in 2001. Spammers, virus writers, phishermen, money launderers, and spies now trade busily with each other in a lively online criminal economy—and as they specialize, they get better. New applications, from search to social networks to electronic voting machines, provide new targets. And terrorism has changed the world. In this indispensable, fully updated guide, Ross Anderson reveals how to build systems that stay dependable whether faced with error or malice.

  Here's straight talk about

  - Technical engineering basics—cryptography, protocols, access controls, and distributed systems
  - Types of attack—phishing, Web exploits, card fraud, hardware hacks, and electronic warfare
  - Specialized protection mechanism—what biometrics, seals, smartcards, alarms, and DRM do, and how they fail
  - Security economics—why companies build insecure systems, why it's tough to manage security projects, and how to cope
  - Security psychology—the privacy dilemma, what makes security too hard to use, and why deception will keep increasing
  - Policy—why governments waste money on security, why societies are vulnerable to terrorism, and what to do about it

- Bishop, Matt. Computer Security: Art and Science. Addison-Wesley Professional, 2002. *Abstract from publisher*

  The importance of computer security has increased dramatically during the past few years. Bishop provides a monumental reference for the theory and practice of computer security. This is a textbook intended for use at the advanced undergraduate and introductory graduate levels, non-University training courses, as well as reference and self-study for security professionals. Comprehensive in scope, this covers applied and practical elements, theory, and the reasons for the design of applications and security techniques. Bishop treats the management and engineering issues of computer. Excellent examples of ideas and mechanisms show how disparate techniques and principles are combined (or not) in widely-used systems. Features a distillation of a vast number of conference papers, dissertations and books that have appeared over the years, providing a valuable synthesis. This book is acclaimed for its scope, clear and lucid writing, and its combination of formal and theoretical aspects with real systems, technologies, techniques, and policies.

9.5.2 Secondary

Note: In addition to the following sources, the instructor should consider using papers that cover recent trends in software security.

- The Association for Computing Machinery (ACM) & IEEE Computer Society (IEEE-CS). *Software Engineering Code of Ethics and Professional Practice* (Version 5.2). ACM/IEEE-CS joint task force on Software Engineering Ethics and Professional Practices (SEEPP). http://www.acm.org/about/se-code (1999).
- Dark, Melissa; Harter, Nathan; Morales, Linda; & Garcia, Mario A. "An Information Security Ethics Education Model." *Journal of Computing Sciences in College 23*, 6 (June 2008): 82-88.
- Dowd, Mark; McDonald, John; & Schuh, Justin. The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities. Addison Wesley, 2007.
- Pollice, Gary. *Ethics and Software Development*. http://www.ibm.com/developerworks/rational/library/may06/pollice/index.html (2006).
- Goertzel, Karen Mercedes; Winograd, Theodore; McKinley, Holly Lynne; Oh, Lyndon; Colon, Michael; McGibbon, Thomas; Fedchak, Elaine; & Viennuea. *Software Security Assurance: State-of-the-Art Report (SOAR)*. Information Assurance Technology Analysis Center (IATAC) & Data and Analysis Center for Software (DACS), 2007. http://iac.dtic.mil/iatac/download/security.pdf

9.6 Assignments

Students are assigned reading for each class period where new material is discussed. Assignments are discussed and assigned in the week shown and due the following week. For some weeks, individual system security assurance exercises are assigned, some based on in-class exercises. A major part of the course is a software system security assurance team project.

- Teams will select an existing application and analyze and judge its system security features: what are the threats to the system; what sort of attacks is the system subject to; what sort of countermeasures are there (or should there be) to mitigate system security risks? The teams will carry out the following activities: organize and plan their work; determine a process for carrying out their work; study and analyze the application; identify the key security issues in the application; decide on appropriate measures to address problems; and report their findings and recommendations.
- The instructor will provide the teams with candidate applications. Ross's book [Ross 2008] discusses a number of application types and examples.

9.7 In-Class Activities

In-class activities are described in the suggested schedule below. Most sessions consist of lecture and related discussion of various assurance assessment topics and also include a group exercise. The exercise should take 15 to 30 minutes and be followed by a 15-to-20-minute class discussion. Note that depending on the time available, the number of activities could be increased or decreased.

9.8 Suggested Schedule

The syllabus in Table 8 below defines in-class discussions and other activities that are intended to reinforce lecture material and homework assignments.

Table 8: Syllabus for the System Security Assurance (SSA) Course

Week	Topic	In-Class Activities	Suggested Reading	Assignment
1	**Introduction to System Security Assurance**	• Discussion of course objectives, content, and activities (including team project) • Presentation of an overview of security engineering issues • Discussion of security issues for various examples: a bank, an air force base, a hospital, and the home • Group exercise to identify security risks in a school system. (This is to be done at a high conceptual level, simply indicating the types of risks that exist.)	• [Anderson 2008] Chapter 1 • [Bishop 2002] Chapter 1	

Week	Topic	In-Class Activities	Suggested Reading	Assignment
2	**Attacker Methods 1**	• Discussion of methods attackers can use to damage software and its associated data via weaknesses in the design or coding of the system. Discussion of attack patterns and attack trees • Group exercise based on exercise 1 in Chapter 13 of [Bishop 2003]	• [Anderson 2008] Chapters 2 • [Bishop 2002] Chapter 13, Sections 19.2, 19.3	• Individual exercise: On the basis of class discussion, complete the in-class exercise, providing detail about how one of the risks identified could be mitigated. • Complete and submit team project member information sheet.
3	**Attacker Methods 2**	• Discussion of attacks used to interfere with an application's or system's operations • Group exercise based on exercise 2a in Chapter 22 of [Bishop 2003]	• [Anderson 2008] Chapters 11, 18 • [Bishop 2002] Chapters 22, 23	Individual exercise based on exercise 2 in Chapter 13 of [Bishop 2003]
4	**Threat Analysis 1**	• Discussion of analysis and modeling of the threats to which newly developed or acquired software is most likely to be vulnerable in specific operating environments and domains • Form project teams and provide guidance for team project proposal.	• [Anderson 2008] Chapters 19, 20, 21 • [Bishop 2002] Chapter 23	Individual exercise based on exercises 2b and 2c in Chapter 22 of [Bishop 2003]
5	**Threat Analysis 2**	• Discussion of analysis and modeling of the threats to which existing software is most likely to be vulnerable in specific operating environments and domains • Group exercise based on exercise 5 in Chapter 23 of [Bishop 2003]	• [Anderson 2008] Chapters 4, 11 • [Bishop 2002] Chapter 23	Team project work
6	**Security Countermeasures 1**	• Discussion of countermeasures such as layers, access controls, privileges, intrusion detection, encryption, and coding checklists • Group exercise based on exercise 1 in Chapter 9 of [Bishop 2003]	• [Anderson 2008] Chapters 4, 5 • [Bishop 2002] Chapters 2, 9	Complete and submit team project proposal: team organization, problem need statement, project schedule and deliverables.
7	**Security Countermeasures 2**	• Discussion of countermeasures such as layers, access controls, privileges, intrusion detection, encryption, and coding checklists • Group exercise based on exercise 1 in Chapter 25 of [Bishop 2003]	• [Anderson 2008] Chapters 6, 8 • [Bishop 2002] Chapters 10, 25	• Individual exercise based on exercise 2 in Chapter 9 of [Bishop 2003] • Team project work

Week	Topic	In-Class Activities	Suggested Reading	Assignment
8	**Planning for Access Control and Authentication**	• Discussion of designing and planning for access control and authentication • Group exercise based on exercise 13 in Chapter 12 of [Bishop 2003]	• [Anderson 2008] Chapter 4 • [Bishop 2002] Chapters 12, 15	• Individual exercise based on exercise 2 in Chapter 25 of [Bishop 2003] • Team project work
9	**Safety and Security Risks**	• Discussion of safety and security risks associated with critical infrastructure systems such as power, telecommunications, water, and air-traffic-control systems • Group exercise to identify security risks in a nuclear power plant system. (This is to be done at a high conceptual level, simply indicating the types of risks that exist.)	[Anderson 2008] Chapters 13, 19, 20, 24	Interim team project progress report
10	**Mitigating Security Risks**	• Mitigating risks with gates, locks, guards, and background checks • Group exercise based on exercise 6 in Chapter 15 of [Bishop 2003]	• [Anderson 2008] Chapters 4, 5 • [Bishop 2002] Chapter 15	• On the basis of class discussion in the previous class, complete the in-class exercise, providing detail about how one of the risks identified could be mitigated. • Team project work
11	**Legal and Ethical Issues 1**	• Obligations that people, who are knowledgeable about attack and prevention methods, have to use their abilities, for both legal and ethical reasons • Group exercise concerning Privacy and Surveillance: The Carnivore case (http://computingcases.org/case_materials/case_materials.html)	• [ACM 1999] • [Pollice 2006]	Team project work
12	**Legal and Ethical Issues 2**	• The legal and ethical considerations involved in analyzing historical events and investigations • Class discussion: How does [ACM 1999] relate to software security engineering?	• [ACM 1999] • [Anderson 2008] Chapters 7, 19, 22, 23	Team project work
13	**Project Presentations**	Student teams make a 20-minute presentation about the results of their projects.		Final project report
14	**Final Exam**			

10 Software Assurance Capstone Experience (SACE)

10.1 Catalog Description

This course focuses on the development or modification of a significant software system, employing software assurance knowledge gained from courses throughout the program. The course includes development or modification of requirements, design, implementation, and testing of the system. Deliverables include a project plan requirements specifications; preliminary and detailed designs; code; and test, verification, and validation results. The course culminates with a presentation of the software product to the customer, including a demonstration of its functional and security features.

10.2 Prerequisites

- SOpA course
- ASD1 course
- ASD2 course
- ASD3 course
- AA course
- SSA course

10.3 Corequisites

- AM course
- ASA course

10.4 Expected Student Outcomes

After completing this course, students will be able to

- establish and specify the required/desired level of assurance for a specific software system
- evaluate the capabilities and limitations of technical environments, languages, and tools for assured software
- identify, analyze, and perform software assurance practices that are relevant for the software to be developed
- demonstrate compliance with laws, regulations, standards, and policies that apply to the software product
- analyze the threats to which the software is most likely to be vulnerable in a specific operating environment and domain, including a risk assessment of security vulnerabilities
- develop requirements specification, architecture, and design specifications that satisfy the required/desired level of assurance for a specific software system

- apply methods, techniques, and tools to construct software modules that meet the functionality and security requirements and implement the modules' security architecture and design specifications
- apply testing and review methods, develop plans, and analyze results that demonstrate that a software product satisfies its functionality and security requirements
- plan for and ensure that the software responds effectively to operational software accidents, failures, and intrusions

10.5 List of Topics

Because of the nature of this course, the course topics include a broad spectrum from the MSwA2010 BoK (Appendix A):

- overview of system security engineering
- project team process, organization, communication, and assessment
- project management (planning, risk management, configuration management) and software assurance plans
- quality assurance, software assurance metrics, and software assurance analytics
- requirements, design, and implementation
- testing, inspections, and reviews, including independent assurance testing
- creation and maintenance of auditable evidence for software assurance
- evolution and operation issues
- project reports and presentations

10.6 Sources

No specific sources are specified for this course; however, students are expected to consult, as needed, the sources used in the prerequisite and corequisite courses.

10.7 Project Guidance

- This is a project-oriented course based on the previous knowledge and experience gained in other program courses. No, or very little, new material will be presented in lectures or class discussions. Any new knowledge or capability needed to complete the project (e.g., domain knowledge or knowledge about the use of a new tool or method) will require research by the students on the project team.
- The project should involve significant software security elements, and software assurance methods and activities should be used in project work. For example, a security risk assessment and an operational plan focused on security risks would be useful deliverables.
- An evolutionary or maintenance project dealing with an existing product would be a good choice for this course. It might be appropriate to do a security assessment of an existing product or service and make needed updates to reduce vulnerabilities.

- There is much debate about the source of project work. Should it be a real-world project with a real customer or a made-up, but realistic, project? Each has advantages and disadvantages. If a real customer is not available or appropriate, the instructor or another faculty member may act as a pseudo customer.
- The instructor typically acts as a team coach or mentor.
- The chief source for assessment for this course would be project artifacts such as the following: process and plan documents; risk and configuration management plans and reports; reports on software assurance audits; requirements and design documents; source code; test plans and reports; inspection and review reports; and team process and product assessments.
- An individual assessment can be based on self-assessment and peer assessment and, if possible, on the quality of an artifact for which the individual had primary responsibility. A teacher might also use individual observations (e.g., how well an individual participates in a team design review) and/or an interview with the individual student.
- If the project has an outside customer, feedback from the customer can be helpful in assessing individual and team performance.

Appendix A: MSwA2010 Body of Knowledge (BoK)

This section describes the MSwA2010 BoK, the core body of knowledge for an MSwA degree. The term *software assurance* used in this section is the expanded definition in Section 2 of *Volume I*. The MSwA2010 BoK includes software assurance practices that are required to support the MSwA2010 outcomes. All software assurance professionals must know these practices to perform their jobs effectively. The MSwA2010 BoK is structured into seven knowledge areas, with each knowledge area subdivided into a set of knowledge units.

The MSwA2010 BoK does not provide detailed descriptions but rather serves as a guide to the body of knowledge by referencing literature that explains and elaborates on the elements (see Appendix B of *Volume I*).

The following knowledge areas are defined in terms of the Bloom cognitive levels, which are described in Appendix A of *Volume I*. Brief descriptions of the outcomes are included for each knowledge area. For detailed descriptions of the outcomes, refer to Section 4 of *Volume I*.

1. Assurance Across Life Cycles

Outcome: Graduates will have the ability to incorporate assurance technologies and methods into life-cycle processes and development models for new or evolutionary system development, and for system or service acquisition.

1.1. Software Life-Cycle Processes

1.1.1. New development (Bloom Level C)

Processes associated with the full development of a software system

1.1.2. Integration, assembly, and deployment (Bloom Level C)

Processes concerned with the final phases of the development of a new or modified software system

1.1.3. Operation and evolution (Bloom Level C)

Processes that guide the operation of the software product and its change over time

1.1.4. Acquisition, supply, and service (Bloom Level C)

Processes that support acquisition, supply, or service of a software system

1.2. Software Assurance Processes and Practices

1.2.1. Process and practice assessment (Bloom Level AP)

Methods, procedures, and tools used to assess assurance processes and practices

1.2.2. Software assurance integration into SDLC phases (Bloom Level AP)

Integration of assurance practices into typical life-cycle phases (for example, requirements engineering, architecture and design, coding, test, evolution, acquisition, and retirement)

2. Risk Management

Outcome: Graduates will have the ability to perform risk analysis and tradeoff assessment, and to prioritize security measures.

2.1. Risk Management Concepts

2.1.1. Types and classification (Bloom Level C)

Different classes of risks (for example, business, project, technical)

2.1.2. Probability, impact, severity (Bloom Level C)

Basic elements of risk analysis

2.1.3. Models, processes, metrics (Bloom Level C)

Models, process, and metrics used in risk management

2.2. Risk Management Process

2.2.1. Identification (Bloom Level AP)

Identification and classification of risks associated with a project

2.2.2. Analysis (Bloom Level AP)

Analysis of the likelihood, impact, and severity of each identified risk

2.2.3. Planning (Bloom Level AP)

Risk management plan covering risk avoidance and mitigation

2.2.4. Monitoring and management (Bloom Level AP)

Assessment and monitoring of risk occurrence and management of risk mitigation

2.3. Software Assurance Risk Management

2.3.1. Vulnerability and threat identification (Bloom Level AP)

Application of risk analysis techniques to vulnerability and threat risks

2.3.2. Analysis of software assurance risks (Bloom Level AP)

Analysis of risks for both new and existing systems

2.3.3. Software assurance risk mitigation (Bloom Level AP)

Plan for and mitigation of software assurance risks

2.3.4. Assessment of Software Assurance Processes and Practices (Bloom Level AP)

As part of risk avoidance and mitigation, assessment of the identification and use of appropriate software assurance processes and practices

3. Assurance Assessment

Outcome: Graduates will have the ability to analyze and validate the effectiveness of assurance operations and create auditable evidence of security measures.

3.1. Assurance Assessment Concepts

3.1.1. Baseline level of assurance; allowable tolerances, if quantitative (Bloom Level AP)

Establishment and specification of the required or desired level of assurance for a specific software application, set of applications, or software-reliant system (and tolerance for same)

3.1.2. Assessment methods (Bloom Level C)

Knowledge of how various methods (such as validation of security requirements, risk analysis, threat analysis, vulnerability assessments and scans, and assurance evidence) can be used to determine if the software/system being assessed is sufficiently secure within tolerances

3.2. Measurement for Assessing Assurance

3.2.1. Product and process measures by life-cycle phase (Bloom Level AP)

Definition and development of key product and process measurements that can be used to validate the required level of software assurance appropriate to a given life-cycle phase

3.2.2. Other performance indicators that test for the baseline as defined in 3.1.1., by life-cycle phase (Bloom Level AP)

Definition and development of additional performance indicators that can be used to validate the required level of software assurance appropriate to a given life-cycle phase

3.2.3. Measurement processes and frameworks (Bloom Level C)

Knowledge of the range of software assurance measurement processes and frameworks and how these might be used to accomplish software assurance integration into SDLC phases

3.2.4. Business survivability and operational continuity (Bloom Level AP)

Definition and development of performance indicators that can specifically address the software/system's ability to meet business survivability and operational continuity requirements, to the extent the software affects these

3.3. Assurance Assessment Process (collect and report measures that demonstrate the baseline as defined in 3.1.1.)

3.3.1. Comparison of selected measurements to the established baseline (Bloom Level AP)

Analysis of key product and process measures and performance indicators to determine if they are within tolerance when compared to the defined baseline

3.3.2. Identification of out-of-tolerance variances (Bloom Level AP)

Identification of measures that are out of tolerance when compared to the defined baselines and ability to develop actions to reduce the variance

4. Assurance Management

Outcome: Graduates will have the ability to make a business case for software assurance, lead assurance efforts, understand standards, comply with regulations, plan for business continuity, and keep current in security technologies.

4.1. Making the Business Case for Assurance

4.1.1. Valuation and cost-benefit models and cost and loss avoidance (Bloom Level AP)

Application of financially based approaches, methods, models, and tools to develop and communicate compelling cost-benefit arguments in support of deploying software assurance practices

4.1.2. Risk analysis (Bloom Level C)

Knowledge of how risk analysis can be used to develop cost-benefit arguments in support of deploying software assurance practices

4.1.3. Compliance justification (Bloom Level C)

Knowledge of how compliance with laws, regulations, standards, and policies can be used to develop cost-benefit arguments in support of deploying software assurance practices

4.1.4. Business impact/needs analysis (Bloom Level C)

Knowledge of how business impact and needs analysis can be used to develop cost-benefit arguments in support of deploying software assurance practices, specifically in support of business continuity and survivability

4.2. Managing Assurance

4.2.1. Project management across the life cycle (Bloom Level C)

Knowledge of how to lead software and system assurance efforts as an extension of normal software development (and acquisition) project management skills

4.2.2. Integration of other knowledge units (Bloom Level AN)

Identification, analysis, and selection of software assurance practices from any knowledge units that are relevant for a specific software development or acquisition project

4.3. Compliance Considerations for Assurance

4.3.1. Laws and regulations (Bloom Level C)

Knowledge of the extent to which selected laws and regulations are relevant for a specific software development or acquisition project, and how compliance might be demonstrated

4.3.2. Standards (Bloom Level C)

Knowledge of the extent to which selected standards are relevant for a specific software development or acquisition project, and how compliance might be demonstrated

4.3.3. Policies (Bloom Level C)

Knowledge of how to develop, deploy, and use organizational policies to accelerate the adoption of software assurance practices, and how compliance might be demonstrated

5. System Security Assurance

Outcome: Graduates will have the ability to incorporate effective security technologies and methods into new and existing systems.

5.1. For Newly Developed and Acquired Software for Diverse Systems

5.1.1. Security and safety aspects of computer-intensive critical infrastructure (Bloom Level K)

Knowledge of safety and security risks associated with critical infrastructure systems such as found, for example, in banking and finance, energy production and distribution, telecommunications, and transportation systems

5.1.2. Potential attack methods (Bloom Level C)

Knowledge of the variety of methods by which attackers can damage software or data associated with that software by exploiting weaknesses in the system design or implementation

5.1.3. Analysis of threats to software (Bloom Level AP)

Analysis of the threats to which software is most likely to be vulnerable in specific operating environments and domains

5.1.4. Methods of defense (Bloom Level AP)

Familiarity with appropriate countermeasures such as layers, access controls, privileges, intrusion detection, encryption, and code review checklists

5.2. For Diverse Operational (Existing) Systems

5.2.1. Historic and potential operational attack methods (Bloom Level C)

Knowledge of and ability to duplicate the attacks that have been used to interfere with an application's or system's operations

5.2.2. Analysis of threats to operational environments (Bloom Level AN)

Analysis of the threats to which software is most likely to be vulnerable in specific operating environments and domains

5.2.3. Design of and plan for access control, privileges, and authentication (Bloom Level AP)

Design of and plan for access control and authentication

5.2.4. Security methods for physical and personnel environments (Bloom Level AP)

Knowledge of how physical access restrictions, guards, background checks, and personnel monitoring can address risks

5.3. Ethics and Integrity in Creation, Acquisition, and Operation of Software Systems

5.3.1. Overview of ethics, code of ethics, and legal constraints (Bloom Level C)

Knowledge of how people who are knowledgeable about attack and prevention methods are obligated to use their abilities, both legally and ethically, referencing the Software Engineering Code of Ethical and Professional Conduct [ACM 2009]

5.3.2. Computer attack case studies (Bloom Level C)

Knowledge of the legal and ethical considerations involved in analyzing a variety of historical events and investigations

6. System Functionality Assurance

Outcome: Graduates will have the ability to verify new and existing software system functionality for conformance to requirements and to help reveal malicious content.

6.1. Assurance Technology

6.1.1. Technology evaluation (Bloom Level AN)

Evaluation of capabilities and limitations of technical environments, languages, and tools with respect to creating assured software functionality and security

6.1.2. Technology improvement (Bloom Level AP)

Recommendation of improvements in technology as necessary within project constraints

6.2. Assured Software Development

6.2.1. Development methods (Bloom Level AP)

Rigorous methods for system requirements, specification, architecture, design, implementation, verification, and testing to develop assured software

6.2.2. Quality attributes (Bloom Level C)

Software quality attributes and how to achieve them

6.2.3. Maintenance methods (Bloom Level AP)

Assurance aspects of software maintenance and evolution

6.3. Assured Software Analytics

6.3.1. Systems analysis (Bloom Level AP)

Analysis of system architectures, networks, and databases for assurance properties

6.3.2. Structural analysis (Bloom Level AP)

Structuring the logic of existing software to improve understandability and modifiability

6.3.3. Functional analysis (Bloom Level AP)

Reverse engineering of existing software to determine functionality and security properties

6.3.4. Analysis of methods and tools (Bloom Level C)

Capabilities and limitations of methods and tools for software analysis

6.3.5. Testing for assurance (Bloom Level AN)

Evaluation of testing methods, plans, and results for assuring software

6.3.6. Assurance evidence (Bloom Level AP)

Development of auditable assurance evidence

6.4. Assurance in Acquisition

6.4.1. Assurance of acquired software (Bloom Level AP)

Assurance of software acquired through supply chains,[5] vendors, and open sources, including developing requirements and assuring delivered functionality and security

6.4.2. Assurance of software services (Bloom Level AP)

Development of service level agreements for functionality and security with service providers and for monitoring compliance

7. System Operational Assurance

Outcome: Graduates will have the ability to monitor and assess system operational security and respond to new threats.

7.1. Operational Procedures

7.1.1. Business objectives (Bloom Level C)

Role of business objectives and strategic planning in system assurance

7.1.2. Assurance procedures (Bloom Level AP)

Creation of security policies and procedures for system operations

7.1.3. Assurance training (Bloom Level K)

Selection of training for users and system administrative personnel in secure system operations

7.2. Operational Monitoring

7.2.1. Monitoring technology (Bloom Level C)

Capabilities and limitations of monitoring technologies, and installation and configuration or acquisition of monitors and controls for systems, services, and personnel

[5] For more information about software security supply chain risk, download the SEI report *Evaluating and Mitigating Software Supply Chain Security Risks* [Ellison 2010].

7.2.2. Operational evaluation (Bloom Level AP)

Evaluation of operational monitoring results with respect to system and service functionality and security

7.2.3. Operational maintenance (Bloom Level AP)

Maintenance and evolution of operational systems while preserving assured functionality and security

7.2.4. Malware analysis (Bloom Level AP)

Evaluation of malicious content and application of countermeasures

7.3. System Control

7.3.1. Responses to adverse events (Bloom Level AN)

Plan for and execution of effective responses to operational system accidents, failures, and intrusions

7.3.2. Business survivability (Bloom Level AP)

Maintenance of business survivability and continuity of operations in adverse environments (see also Knowledge Unit 3, Assurance Assessment)

Having a defined set of student prerequisites, established outcomes, a core body of knowledge, and curriculum architecture is necessary but not sufficient. Often the most challenging part of putting a new program or a new track in place is implementation. The next section provides guidelines and recommendations for faculty members to consider when considering starting an MSwA program.

Appendix B: MSwA BoK Topics Covered by Syllabi

The table below indicates which knowledge areas of the MSwA BoK are covered by which courses in this syllabi.

Table 9: MSwA BoK Topics Covered by the Syllabi

Knowledge Areas	Course That Covers This Area
1. Assurance Across Life Cycles	
1.1. Software Life-Cycle Processes	
1.1.1. New development	Assured Software Development 1
1.1.2. Integration, assembly, and deployment	Assured Software Development 1
1.1.3. Operation and evolution	Assured Software Development 1
1.1.4. Acquisition, supply, and service	Assured Software Development 1
1.2. Software Assurance Processes and Practices	
1.2.1. Process and practice assessment	Assured Software Development 1
1.2.2. Software assurance integration into SDLC phases	Assured Software Development 1 Assured Software Development 2
2. Risk Management	
2.1. Risk Management Concepts	Assurance Management
2.1.1. Types and classification	
2.1.2. Probability, impact, severity	
2.1.3. Models, processes, metrics	
2.2. Risk Management Process	Assurance Management
2.2.1. Identification	
2.2.2. Analysis	
2.2.3. Planning	
2.2.4. Monitoring and management	
2.3. Software Assurance Risk Management	Assurance Management
2.3.1. Vulnerability and threat identification	
2.3.2. Analysis of software assurance risks	
2.3.3. Software assurance risk mitigation	
2.3.4. Assessment of Software Assurance Processes and Practices	
3. Assurance Assessment	
3.1. Assurance Assessment Concepts	Assurance Assessment
3.1.1. Baseline level of assurance; allowable tolerances, if quantitative	
3.1.2. Assessment methods	

Knowledge Areas	Course That Covers This Area
3.2. Measurement for Assessing Assurance	Assurance Assessment
3.2.1. Product and process measures by life-cycle phase	
3.2.2. Other performance indicators that test for the baseline, by life-cycle phase	
3.2.3. Measurement processes and frameworks	
3.2.4. Business survivability and operational continuity	
3.3. Assurance Assessment Process (collect and report measures that demonstrate the baseline)	Assurance Assessment
3.3.1. Comparison of selected measurements to the established baseline	
3.3.2. Identification of out-of-tolerance variances	
4. Assurance Management	
4.1. Making the Business Case for Assurance	Assurance Management
4.1.1. Valuation and cost/benefit models, cost and loss avoidance, return on investment	
4.1.2. Risk analysis	
4.1.3. Compliance justification	
4.1.4. Business impact/needs analysis	
4.2. Managing Assurance	Assurance Management
4.2.1. Project management across the life cycle	
4.2.2. Integration of other knowledge units	
4.3. Compliance Considerations for Assurance	Assurance Management System Operational Assurance
4.3.1. Laws and regulations	
4.3.2. Standards	
4.3.3. Policies	
5. System Security Assurance	
5.1. For Newly Developed and Acquired Software for Diverse Applications	System Security Assurance
5.1.1. Security and safety aspect of computer-intensive critical infrastructure	
5.1.2. Potential attack methods	
5.1.3. Analysis of threats to software	
5.1.4. Methods of defense	
5.2. For Diverse Operational (Existing) Systems	System Security Assurance
5.2.1. Historic and potential operational attack methods	
5.2.2. Analysis of threats to operational environments	
5.2.3. Designing of and plan for access control, privileges, and authentication	

Knowledge Areas	Course That Covers This Area
5.2.4. Security methods for physical and personnel environments	
5.3. Ethics and Integrity in Creation, Acquisition, and Operation of Software Systems	System Security Assurance
5.3.1. Overview of ethics, code of ethics, and legal constraints	
5.3.2. Computer attack case studies	
6. System Functionality Assurance	
6.1. Assurance Technology	
6.1.1. Technology evaluation	Assured Software Development 2
6.1.2. Technology improvement	Assured Software Development 1 Assured Software Development 2
6.2. Assured Software Development	Assured Software Analytics Assured Software Development 1 Assured Software Development 3
6.2.1. Development methods	Assured Software Development 1 Assured Software Development 2
6.2.2. Quality attributes	
6.2.3. Maintenance methods	
6.3. Assured Software Analytics	Assured Software Analytics
6.3.1. Systems analysis	
6.3.2. Structural analysis	
6.3.3. Functional analysis	
6.3.4. Analysis of methods and tools	
6.3.5. Testing for assurance	
6.3.6. Assurance evidence	
6.4. Assurance in Acquisition	Assurance Assessment Assured Software Analytics
6.4.1. Assurance of acquired software	
6.4.2. Assurance of software services	
7. System Operational Assurance	
7.1. Operational Procedures	System Operational Assurance
7.1.1. Business objectives	
7.1.2. Assurance procedures	
7.1.3. Assurance training	
7.2. Operational Monitoring	System Operational Assurance
7.2.1. Monitoring technology	
7.2.2. Operational evaluation	

Knowledge Areas	Course That Covers This Area
7.2.3. Operational maintenance	
7.2.4. Malware analysis	
7.3. System Control	System Operational Assurance
7.3.1. Responses to adverse events	
7.3.2. Business survivability	

Appendix C: Acronym List

AA
Assurance Assessment

AM
Assurance Management

ASA
Assured Software Analytics

ASD1
Assured Software Development 1

ASD2
Assured Software Development 2

ASD3
Assured Software Development 3

ATAM®
Architecture Tradeoff Analysis Method®

Bloom Cognitive Levels
K—knowledge
C—comprehension
AP—application
AN—analysis

BoK
body of knowledge

BSI
Build Security In

BSIMM
Building Security In Maturity Model

CAPEC
Common Attack Pattern Enumeration and Classification

CERT/CC
CERT® Coordination Center

® ATAM, Architecture Tradeoff Analysis Method, and CERT are registered trademarks owned by Carnegie Mellon University.

CERT-RMM
CERT® Resilience Management Model

CLASP
Comprehensive, Lightweight Application Security Process

CMMI®
Capability Maturity Model Integration℠

COTS
commercial, off-the-shelf

CP
compliance and policy

CWE
Common Weakness Enumeration

DHS
Department of Homeland Security

GQM
Goal Question Metric

ISO
International Organization for Standardization

MA
measurement and analysis

NIST
National Institute of Standards and Technology

OWASP
Open Web Application Security Project

PT
penetration testing

QAW
Quality Attribute Workshop

SA
static analysis

® CMMI is a registered trademark owned by Carnegie Mellon University.

℠ Capability Maturity Model Integration is a service mark of Carnegie Mellon University.

SACE
Software Assurance Capstone Experience

SAMATE
Software Assurance Metrics and Tools Evaluation

SAMM
Software Assurance Maturity Model

SDL
Security Development Lifecycle

SDLC
software development life cycle

SEI
Software Engineering Institute

SOA
service-oriented architecture

SOpA
System Operational Assurance

SQUARE
Security Quality Requirements Engineering

SSA
System Security Assurance

Bibliography

URLs are valid as of the publication date of this document.

[ACM 1999]
The Association for Computing Machinery (ACM) & IEEE Computer Society (IEEE-CS). *Software Engineering Code of Ethics and Professional Practice* (Version 5.2). ACM/IEEE-CS joint task force on Software Engineering Ethics and Professional Practices (SEEPP). http://www.acm.org/about/se-code (1999).

[Ahern 2008]
Ahern, Dennis M.; Clouse, Aaron; & Turner, Richard. *CMMI Distilled: A Practical Introduction to Integrated Process Improvement*, 3rd ed. Addison-Wesley Professional, 2008.

[Alberts 2010a]
Alberts, Christopher; Allen, Julia; & Stoddard, Robert. *Integrated Measurement and Analysis Framework for Software Security* (CMU/SEI-2010-TN-025). Software Engineering Institute, Carnegie Mellon University, 2010.

[Alberts 2010b]
Alberts, Christopher J. & Dorofee, Audrey J. *Risk Management Framework* (CMU/SEI-2010-TR-071). Carnegie Mellon University, Software Engineering Institute, 2010. http://www.sei.cmu.edu/library/abstracts/reports/10tr017.cfm

[Alexander 2003]
Alexander, Ian. "Misuse Case: Use Cases with Hostile Intent." IEEE Software 20, 1 (January/ February 2003): 58–66.

[Allen 2008]
Allen, Julia H.; Barnum, Sean; Ellison, Robert J.; McGraw, Gary; & Mead, Nancy R. Ch. 4, "Secure Software Architecture and Design," 115-150. *Software Security Engineering: A Guide for Project Managers*. Addison-Wesley Professional, 2008.

[Altran 2010]
Altran Group. *Correctness by Construction*. http://www.altran-praxis.com/cbyc.aspx (2010).

[Anderson 2008]
Anderson, Ross J. *Security Engineering: A Guide to Building Dependable Distributed Systems*, 2nd ed. Wiley, 2008.

[AS/NSZ 2009]
Australian/New Zealand Standard (AS/NZS) & International Organization for Standardization (ISO). AS/NZS ISO 31000: 2009 *Risk Management—Principles and Guidelines*, 1st ed. AS/NZS, November 2009.

[Bagheri 2008]
Bagheri, Hamid & Mirian-Hosseinabadi, Seyed-Hassan. "Injecting Security as Aspectable NFR into Software Architecture," *Proceedings of the 14th Asia-Pacific Software Engineering Conference*. Nagoya, Japan. Dec. 2007. IEEE Computer Society Press, 2008.

[Barbacci 2003]
Barbacci, Mario R.; Ellison, Robert J.; Lattanze, Anthony J.; Stafford, Judith A.; Weinstock, Charles B.; & Wood, William G. *Quality Attribute Workshops (QAWs)*, 3rd ed. (CMU/SEI-2003-TR-016). Software Engineering Institute, Carnegie Mellon University, 2003. http://www.sei.cmu.edu/library/abstracts/reports/03tr016.cfm

[Berg 2005]
Berg, Clifford J. *High Assurance Design: Architecting Secure and Reliable Enterprise Applications*. Addison-Wesley Professional, 2005.

[Bishop 2002]
Bishop, Matt. *Computer Security: Art and Science.* Addison-Wesley Professional, 2002.

[Blanchette 2009]
Blanchette, Stephen, Jr.; Crosson, Steven; & Boehm, Barry. *Evaluating the Software Design of a Complex System of Systems* (CMU/SEI-2009-TR-023, ESC-TR-2009-023). Software Engineering Institute, Carnegie Mellon University, 2009. http://www.sei.cmu.edu/library/abstracts/reports/09tr023.cfm

[Bloom 1956]
Bloom, B. S., ed. Taxonomy of Educational Objectives: The Classification of Educational Goals: Handbook I, Cognitive Domain. Longmans, 1956.

[Bode 2009]
Bode, Stephan; Fischer, Anja; Kühnhauser, Winfried; & Riebisch, Matthias. "Software Architectural Design Meets Security Engineering," *Proceedings of the 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*. San Francisco, CA, April 2009. IEEE Computer Society Press, 2009.

[Caralli 2004]
Caralli, Richard; Stevens, James F.; Bradford, J. Wilke; & Wilson, William R. *The Critical Success Factor Method: Establishing a Foundation for Enterprise Security Management* (CMU/SEI-2004-TR-010). Carnegie Mellon University, Software Engineering Institute, July 2004. http://www.sei.cmu.edu/library/abstracts/reports/04tr010.cfm

[CERT 2008]
CERT. *Insider Threat and the Software Development Life Cycle* (podcast). Software Engineering Institute, Carnegie Mellon University. http://www.cert.org/podcast/show/20080304cappelli.html (2008).

[CERT 2009]
CERT. *Integrating Privacy Practices into the Software Development Life Cycle* (podcast). Software Engineering Institute, Carnegie Mellon University. http://www.cert.org/podcast/show/20091222hood.html (2009).

[CERT 2010a]
CERT. *CERT Secure Coding Standards*. https://www.securecoding.cert.org/ (2010).

[CERT 2010b]
CERT. *CERT Resilience Management Model*. http://www.cert.org/resilience/rmm.html (2010).

[CERT 2010c]
CERT. *SQUARE* (educational materials for download). Software Engineering Institute, Carnegie Mellon University. http://www.cert.org/sse/square.html (2010).

[CERT 2011]
CERT. *Homepage*. http://www.cert.org/ (2011).

[CloudFail 2011]
CloudFail.net. *Homepage*. http://cloudfail.net/ (2011).

[CMMI 2007]
CMMI Product Team. *CMMI for Acquisition. Version 1.2* (CMU/SEI-2007-TR-017, ESC-TR-2007-017). Software Engineering Institute, Carnegie Mellon University, 2007. http://www.sei.cmu.edu/library/abstracts/reports/07tr017.cfm

[CMMI 2009]
CMMI Product Team. *CMMI for Services, Version 1.2* (CMU/SEI-2009-TR-001, ESC-TR-2009-001). Software Engineering Institute, Carnegie Mellon University, 2009. http://www.sei.cmu.edu/library/abstracts/reports/09tr001.cfm

[CMMI 2010]
CMMI Product Team. *CMMI for Development, Version 1.3* (CMU/SEI-2010-TR-033). Carnegie Mellon University, Software Engineering Institute, November 2010. http://www.sei.cmu.edu/library/abstracts/reports/10tr033.cfm

[CoFI 2008]
CoFI. *CASL from CoFI*. http://www.informatik.uni-bremen.de/cofi/wiki/index.php/CASL (2008).

[Committee 2010]
Committee for Advancing Software-Intensive Systems Producibility; Computer Science and Telecommunications Board; & Division on Engineering and Physical Sciences. *Critical Code: Software Producibility for Defense*. National Academy of Sciences, 2010.

[Community 2007]
Community Z Tools. *Overview*. http://czt.sourceforge.net/ (2007).

[Dark 2008]
Dark, Melissa; Harter, Nathan; Morales, Linda; & Garcia, Mario A. "An Information Security Ethics Education Model." *Journal of Computing Sciences in College 23*, 6 (June 2008): 82-88.

[DHS 2008]
Department of Homeland Security (DHS) Software Assurance (SwA) Acquisition Working Group. *Software Assurance in Acquisition: Mitigating Risks to the Enterprise*. https://buildsecurityin.us-cert.gov/swa/downloads/SwA_in_Acquisition_102208.pdf (2008).

[DHS 2008–2009a]
Department of Homeland Security (DHS). *Build Security In, Best Practices*. https://buildsecurityin.us-cert.gov/bsi/articles/best-practices.html (2008–2009).

[DHS 2008–2009b]
Department of Homeland Security (DHS). *Build Security In, Secure Software Development Life Cycle (SDLC) Process* (articles). https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/sdlc.html (2008-2009).

[DHS 2010]
Department of Homeland Security (DHS). *Security Requirements Engineering* (articles). https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/requirements.html (2010).

[Dowd 2007]
Dowd, Mark; McDonald, John; & Schuh, Justin. *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities.* Addison Wesley, 2007.

[Eagle 2008]
Eagle, Chris. *The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler*. No Starch Press, 2008.

[Ellison 2010a]
Ellison, Robert J.; Goodenough, John B.; Weinstock, Charles B.; & Woody, Carol. *Evaluating and Mitigating Software Supply Chain Security Risks* (CMU/SEI-2010-TN-016). Software Engineering Institute, Carnegie Mellon University, 2010. http://www.sei.cmu.edu/library/abstracts/reports/10tn016.cfm

[Ellison 2010b]
Ellison, Robert J. & Woody, Carol. *Considering Software Supply Chain Risks*. https://buildsecurityin.us-cert.gov/bsi/resources/1185-BSI/1207-BSI.html (2010).

[Epstein 2006]
Epstein, Jeremy; Matsumoto, Scott; & McGraw. "Software Security and SOA: Danger, Will Robinson!" *IEEE Security & Practice 4*, 1 (January/February 2006): 80–83.

[Fitzgerald 2010]
Fitzgerald, John. *Welcome to the VDM Portal*. http://www.vdmportal.org/twiki/bin/view (2010).

[Garcia 2006]
Garcia, Suzanne & Turner, Richard. *CMMI Survival Guide: Just Enough Process Improvement*. Addison-Wesley Professional, 2006.

[Gerhart 2002]
Gerhart, Susan; Hogle, Jan; & Crandall, Jedidiah. *How Do Buffer Overflow Attacks Work?* http://nsfsecurity.pr.erau.edu/bom/ (2002).

[Goertzel 2007]
Goertzel, Karen Mercedes; Winograd, Theodore; McKinley, Holly Lynne; Oh, Lyndon; Colon, Michael; McGibbon, Thomas; Fedchak, Elaine; & Viennuea. *Software Security Assurance: State-of-the-Art Report (SOAR)*. Information Assurance Technology Analysis Center (IATAC) & Data and Analysis Center for Software (DACS), 2007. http://iac.dtic.mil/iatac/download/security.pdf

[Gold 2004]
Gold, Nicolas; Mohan, Andrew; Knight, Claire; & Munro, Malcolm. "Understanding Service-Oriented Software," *IEEE Software 21*, 2 (March/April 2004): 71–77.

[Golze 2008]
Golze, Andreas; Sarbiewski, Mark; & Zahm, Alain. *Optimize Quality for Business Outcomes: A Practical Approach to Software Testing*. Wiley Publishing, 2008.

[Graham 2006]
Graham, Dan. *Introduction to the CLASP Process*. https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/requirements/548-BSI.html (2006).

[Grembi 2009]
Grembi, Jason. "Secure Software Development - A Security Programmer's Guide." Tutorial at *11th Semi-Annual Software Assurance Forum*. Arlington, VA, November 2009. Software Engineering Institute, Carnegie Mellon University, 2009. https://www.vte.cert.org/vteweb/go/2699.aspx

[Haley 2008]
Haley, Charles B; Laney, Robin; Moffett, Jonathan D.; & Nuseibeh, Bashar. Ch. 214, "Arguing Satisfaction of Security Requirements." *Information Security and Ethics: Concepts, Methodologies, Tools, and Applications*. 6 vols. Idea Group Reference, 2008.

[Hansson 2010]
Hansson, Jörgen; Wrage, Lutz; Feiller, Peter H.; & Morley, Johhn. "Architectural Modeling to Verify Security and Nonfunctional Behavior." *IEEE Security & Practice 8*, 1: (Jan./Feb. 2010): 43–49.

[Holt 2010]
Holt, Alan & Huang, Chi-Yu. *802.11 Wireless Networks: Security and Analysis*. Springer, 2010.

[Howard 2003]
Howard, Michael & LeBlanc, David. *Writing Secure Code*, 2nd ed. Microsoft Press, 2003.

[Howard 2005]
Howard, Michael; LeBlanc, David; & Viega, John. *19 Deadly Sins of Software Security*. McGraw-Hill, 2005.

[Howard 2006]
Howard, Michael & Lipner, Steve. *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software*. Microsoft Press, 2006.

[IBM 2009]
IBM. *IBM Point of View: Security and Cloud Computing.* http://www.ibm.com/common/ssi/fcgi-bin/ssialias?infotype=SA&subtype=WH&appname=SWGE_TI_SE_USEN&htmlfid=TIW14045USEN&attachment=TIW14045USEN_HR.PDF (2009).

[IBM 2011]
IBM. *Requirements Management and Definition*. http://www-01.ibm.com/software/rational/offerings/lifecycle/ (2011).

[IEEE 1998a]
IEEE Standards Association (IEEE-SA). IEEE Std 1061–1998 *IEEE Standard for a Software Quality Metrics Methodology*. IEEE–SA, 1998.

[IEEE 1998b]
IEEE Standards Association (IEEE–SA) IEEE Std 1219-1998 *IEEE Standard for Software Maintenance*. IEEE–SA, 1998.

[IEEE 1998c]
IEEE Standards Association (IEEE–SA). IEEE Std 1062–1998 *IEEE Recommended Practice for Software Acquisition*. IEEE–SA, 1998.

[IEEE 2007]
IEEE Standards Association (IEEE–SA). IEEE Std 15939–2007 *IEEE Systems and Software Engineering—Measurement Process*. IEEE–SA, 2007.

[Ingalsbe 2008]
Ingalsbe, Jeffrey A.; Kunimatsu, Louis; Baeten, Tim; & Mead, Nancy R. “Threat Modeling: Diving into the Deep End.” *IEEE Software* 25, 1 (January/February 2008). https://buildsecurityin.us-cert.gov/bsi/resources/articles/932-BSI.html

[ISO 2005]
International Organization for Standardization and International Electrotechnical Commission (ISO/IEC). *ISO/IEC 27002:2005 Information Technology – Security Techniques – Code of Practice for Information Security Management*. ISO/IEC, 2005.

[ISO 2007]
International Organization for Standardization (ISO). *ISO/IEC 15939:2007 Systems and Software Engineering—Measurement Process*. ISO, 2007.

[ISO 2008]
International Organization for Standardization (ISO). *ISO/IEC FCD 27005: 2008 Information Technology—Security Techniques—Information Security Risk Management*, 2nd ed. ISO, June 2008.

[Jacky 1996]
Jacky, Jonathan. *The Way of Z: Practical Programming with Formal Methods*. Cambridge University Press, 1996.

[Joint Task Force 2009]
Joint Task Force Transformation Initiative. *Recommended Security Controls for Federal Information Systems and Organizations* (NIST Special Publication 800-53), Revision 3. National Institute of Standards and Technology, August 2009. Updated May 2010. http://csrc.nist.gov/publications/nistpubs/800-53-Rev3/sp800-53-rev3-final_updated-errata_05-01-2010.pdf

[Joint Task Force 2010]
Joint Task Force Transformation Initiative. *Guide for Applying the Risk Management Framework to Federal Information Systems* (NIST Special Publication 800-37), Revision 1. National Institute of Standards and Technology, February 2010. http://csrc.nist.gov/publications/nistpubs/800-37-rev1/sp800-37-rev1-final.pdf

[Kan 2002]
Kan, Stephen H. *Metrics and Models in Software Quality Engineering*, 2nd ed. Addison-Wesley, 2002.

[Kazman 2000]
Kazman, Rick; Klein, Mark H.; & Clements, Paul C. *ATAM: Method for Architecture Evaluation* (CMU/SEI-2000-TR-004). Software Engineering Institute, Carnegie Mellon University, 2000. http://www.sei.cmu.edu/library/abstracts/reports/00tr004.cfm

[Killcrece 2008]
Killcrece, Georgia. *Incident Management*. https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/incident/223-BSI.html (2005-2008).

[Leroy 2003]
Leroy, Xavier. "Computer Security from a Programming Language and Static Analysis Perspective," 1–9. *Proceedings of the 12th European Conference on Programming*. Warsaw, Poland, April 2003. Springer-Verlag, 2003.

[Linger 1979]
Linger, R.; Mills, H.; & Witt, B. *Structured Programming: Theory and Practice.* Addison-Wesley, 1979.

[Mansourov 2011]
Mansourov, Nicolai & Campara, Djenana. *System Assurance: Beyond Detecting Vulnerabilities.* Elsevier, 2011. http://www.elsevierdirect.com/ISBN/9780123814142/System-Assurance

[Massacci 2007]
Massacci, Fabio; Mylopoulos, John; & Zannone, Nicola. "Computer-Aided Support for Secure Tropos." *Automated Software Engineering 14*, 3 (September 2007): 341–364.

[McGraw 2010]
McGraw, Gary; Chess, Brian; & Migues, Sammy. *Building Security In Maturity Model (BSIMM)*. http://bsimm.com/ (2010).

[Mead 2008]
Mead, Nancy R. *The Common Criteria*. https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/requirements/239-BSI.html (2008).

[Mead 2009]
Mead, Nancy R.; Allen, Julia H.; Conklin, W. Arthur; Drommi, Antonio; Harrison, John; Ingalsbe, Jeff; Rainey, James; & Shoemaker, Dan. *Making the Business Case for Software Assurance* (CMU/SEI-2009-SR-001). Software Engineering Institute, Carnegie Mellon University, 2009. http://www.sei.cmu.edu/library/abstracts/reports/09sr001.cfm

[Mead 2010]
Mead, Nancy R.; Allen, Julia H.; Ardis, Mark; Hilburn, Thomas B.; Kornecki, Andrew J.; Linger, Rick; & McDonald, James. *Software Assurance Curriculum Project Volume I: Master of Software Assurance Reference Curriculum* (CMU/SEI-2010-TR-005). Software Engineering Institute, Carnegie Mellon University, 2010. http://www.sei.cmu.edu/library/abstracts/reports/10tr005.cfm

[Mell 2005]
Mell, Peter; Kent, Karen; & Nusbaum, Joseph. *Guide to Malware Incident Prevention and Handling* (NIST Special Publication 800-83). National Institute of Standards and Technology, November 2005. http://csrc.nist.gov/publications/nistpubs/800-83/SP800-83.pdf

[Mellado 2010]
Mellado, Daniel; Fernández-Medina, Eduardo; & Piattini, Mario. "A Comparison of Software Design Security Metrics," 236–242. *Proceedings of the Fourth European Conference on Software Architecture*. Copenhagen, Denmark, Aug. 2010. ACM, 2010.

[Merkow 2010]
Merkow, Mark S. & Raghavan, Lakshmikanth. *Secure and Resilient Software Development*. CRC Press, 2010.

[Miller 2007]
Miller, Barton P.; Cooksey, Gregory; & Moore, Fredrick. "An Empirical Study of the Robustness of MacOS Applications Using Random Testing." *ACM SIGOPS Operating Systems Review 41*, 1 (January 2007): 78-86.

[MITRE 2010a]
The MITRE Corporation (MITRE). *CAPEC: Common Attack Pattern Enumeration and Classification*. http://capec.mitre.org/ (2010).

[MITRE 2010b]
The MITRE Corporation (MITRE). *Common Weakness Enumeration*. http://cwe.mitre.org/ (2010).

[Mouratidis 2007]
Mouratidis, Haralambos & Giorgini, Paolo. *Integrating Security and Software Engineering: Advances and Future Visions*. Idea Group Publishing, 2007.

[Mouratidis 2010]
Mouratidis, Haralambos & Jurjens, Jan. "From Goal-Driven Security Requirements Engineering to Secure Design." *International Journal of Intelligent Systems 25*, 8 (August 2010): 813–840.

[Myers 2006]
Myers, Andrew. *PLD'06 Tutorial T1: Enforcing and Expressing Security with Programming Languages*. http://www.cs.cornell.edu/andru/pldi06-tutorial/06jun-pldi-tutorial.pdf (2006).

[NIST 2005]
National Institute of Standards and Technology (NIST). *SAMATE: Software Assurance Metrics and Tool Evaluation*. http://samate.nist.gov/Main_Page.html (2005).

[Open Group 2010]
The Open Group. *TOGAF*. http://www.opengroup.org/togaf/ (2010).

[OpenSAMM Project 2009]
OpenSAMM Project. *Software Assurance Maturity Model (SAMM) v1.0*. http://www.owasp.org/index.php/Category:Software_Assurance_Maturity_Model (2009).

[Pollice 2006]
Pollice, Gary. *Ethics and Software Development*. http://www.ibm.com/developerworks/rational/library/may06/pollice/index.html (2006).

[Pressman 2009]
Pressman, Roger S., *Software Engineering: A Practitioner's Approach*, 6th ed. McGraw Hill, 2009.

[Ray 2006]
Ray, Arnab & Cleaveland, Rance. "A Software Architectural Approach to Security By Design," *Proceedings of the 30th International Computer Software and Applications Conference*. Chicago, Ill, Sept. 2006. IEEE Computer Society Press, 2006.

[Rehman 2009]
Rehman, S. & Mustafa, K. "Research on Software Design Level Security Vulnerabilities." *ACM SIGSOFT Software Engineering Notes 34*, 6 (November 2009): 1–5.

[Ross 2008]
Ross, Ron; Katzke, Stu; Johnson, Arnold; Swanson, Marianne; & Stoneburner, Gary. *Managing Risk from Information Systems: An Organizational Perspective* (NIST Special Publication 800-39), 2nd draft. National Institute of Standards and Technology, April 2008. http://www.smartgridinformation.info/pdf/2283_doc_1.pdf

[SAFECode 2008a]
Software Assurance Forum for Excellence in Code (SAFECode). *Software Assurance: An Overview of Current Industry Best Practices*. http://www.safecode.org/publications/SAFECode_BestPractices0208.pdf (2008).

[SAFECode 2008b]
SAFECode. *Fundamental Practices for Software Development: A Guide to the Most Effective Secure Development Practices in Use Today*. http://www.safecode.org/publications/SAFECode_Dev_Practices1108.pdf (2008).

[SANS 2011]
The SANS Institute. *Introduction to the SANS Security Policy Project*. http://www.sans.org/security-resources/policies/ (2011).

[Scarfone 2008]
Scarfone, Karen; Grance, Tim; & Masone, Kelly. *Computer Security Incident Handling Guide* (NIST Special Publication 800-61), Revision 1. National Institute of Standards and Technology, March 2008. http://csrc.nist.gov/publications/nistpubs/800-61-rev1/SP800-61rev1.pdf

[Schumacher 2006]
Schumacher, Markus; Fernandez-Buglioni, Eduardo; Hybertson, Duane; Buschmann, Frank; & Sommerlad, Peter. *Security Patterns: Integrating Security and Systems Engineering*, Wiley Series in Software Design Patterns, 2006.

[Seacord 2005]
Seacord, Robert C. *Secure Coding in C and C++*. Addison-Wesley, 2005. http://www.sei.cmu.edu/library/abstracts/books/0321335724.cfm

[Stoneburner 2001]
Stoneburner, Gary; Hayden, Clark; & Feringa, Alexis. *Engineering Principles for Information Technology Security (A Baseline for Achieving Security).* National Institute of Standards and Technology (NIST), 2001.

[Swanson 2010]
Swanson, Marianne; Bowen, Pauline; Phillips, Amy Wohl; Gallup, Dean; & Lynes, David. *Contingency Planning Guide for Federal Information Systems* (NIST Special Publication 800-34), Revision 1. National Institute of Standards and Technology, May 2010. http://csrc.nist.gov/publications/nistpubs/800-34-rev1/sp800-34-rev1_errata-Nov11-2010.pdf

[SWIFT System 2010]
SWIFT System. *Swift: making web applications secure by construction*. http://www.cs.cornell.edu/jif/swift/ (2010).

[Thiagarajan 2003]
Thiagarajan, Val. *Information Security Management: BS 7799.2:2002: Audit Check List for SANS*. http://www.sans.org/score/checklists/ISO_17799_checklist.pdf (2003).

[Walton 2009]
Walton, G.; Linger, R.; and Longstaff, T. "Computational Evaluation of Software Security Attributes," 1–10. *Proceedings of the 42nd Hawaii International Conference on System Sciences*. Los Alimitos, CA, Jan. 2009. IEEE Computer Society Press, 2009.

[Wikipedia 2011]
Wikipedia. *Fagan Inspection*. http://en.wikipedia.org/wiki/Fagan_inspection (2011).

[Wysopal 2006]
Wysopal, Chris; Nelson, Lucas; Dai Zovi, Dino; & Dustin, Elfriede. *The Art of Software Security Testing*: *Identifying Software Security Flaws.* Addison-Wesley Professional, 2006.

[Zannone 2009]
Zannone, Nicola. "The Si* Modeling Framework: Metamodel and Applications." *International Journal of Software Engineering and Knowledge Engineering 19*, 5 (August 2009): 727–746.

[ZETA System 2010]
The ZETA System. *Overview*. http://uebb.cs.tu-berlin.de/zeta/ (2010).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	March 2011; Revised July 2011	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Software Assurance Curriculum Project Volume III: Master of Software Assurance Course Syllabi	FA8721-05-C-0003

6. AUTHOR(S)

Nancy R. Mead, Julia H. Allen, Mark Ardis (Stevens Institute of Technology), Thomas B. Hilburn (Embry-Riddle Aeronautical University), Andrew J. Kornecki (Embry-Riddle Aeronautical University), & Richard C. Linger (Oak Ridge National Laboratory)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2011-TR-013

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	ESC-TR-2011-013

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

Modern society depends on software systems of ever-increasing scope and complexity in virtually every sphere of human activity including business, finance, energy, transportation, education, communication, government, and defense. Because the consequences of failure can be severe, dependable functionality and security are essential. As a result, software assurance is emerging as an important discipline for the development, acquisition, and operation of software systems and services that provide requisite levels of dependability and security.

This report, the third volume in the Software Assurance Curriculum Project sponsored by the U.S. Department of Homeland Security, provides sample syllabi for the nine core courses in the Master of Software Assurance Reference Curriculum. That curriculum, detailed in Volume I, Master of Software Assurance Reference Curriculum (CMU/SEI-2010-TR-005), presents a body of knowledge from which to create a Master of Software Assurance degree program, as both a stand-alone offering and as a track within existing software engineering and computer science master's degree programs. Volume II, Undergraduate Course Outlines (CMU/SEI-2010-TR-019), presents seven course outlines that could be used in an undergraduate curriculum specialization for software assurance.

This volume is part of our transition plan for assisting educators who wish to implement a Master of Software Assurance degree program, specialization, or certificate program. In addition to application in a standard university program, these syllabi may also be useful for educators developing courses for industry practitioners.

14. SUBJECT TERMS	15. NUMBER OF PAGES
software security engineering, software assurance education, software assurance curriculum	117

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL